Universidad de Zaragoza

Facultad de Ciencias

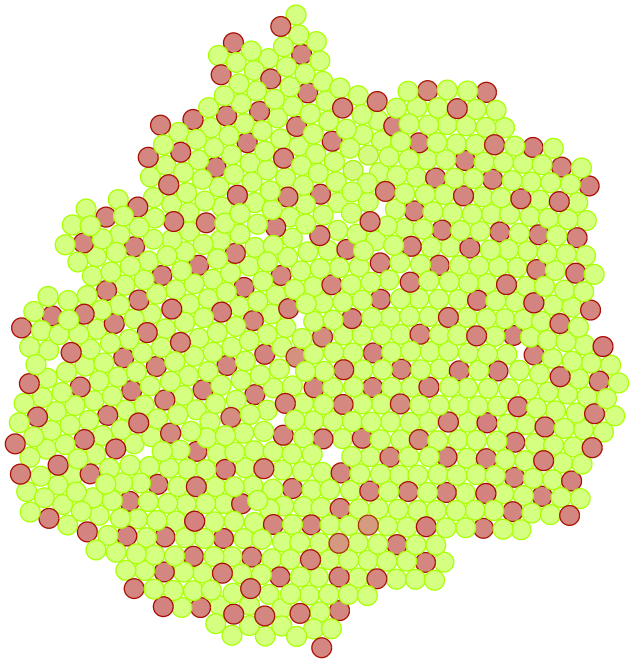
Departamento de Fsica de la Materia Condensada

Trabajo Fin de Grado (Física)

Sistemas dinámicos y expresión de genes: cómo se forman los patrones celulares

Autor:
Pablo Javier Blasco Hernández

Directores:
Dr. Fernando Falo
Dr. Pierpaolo Bruscolini



Índice

Mathematical reasoning may be regarded rather schematically as the exercise of a combination of two facilities, which we may call intuition and ingenuity. The activity of the intuition consists in making spontaneous judgements which are not the result of conscious trains of reasoning... The exercise of ingenuity in mathematics consists in aiding the intuition through suitable arrangements of propositions, and perhaps geometrical figures or drawings.

Alan Turing,
Systems of Logic Based on Ordinals
(1938)

1. Introducción

La naturaleza muestra cotidianamente ejemplos de estructuras (ordenadas o no) que van desde escalas cosmológicas (como son las estructuras de las galaxias) a las escalas más pequeñas, en el mundo nanoscópico y biológico. Dichas estructuras son inhomogeneidades en el espacio y el tiempo. Cabe preguntarse cómo surgen, pues la termodinámica del equilibrio describe estados homogéneos donde la difusión *borra* la localización de fluctuaciones e impide que sean la semilla para la formación de estructuras. Sin embargo, la realidad es que el universo es un sistema que está fuera del equilibrio y ello permite que existan flujos de energía y materia que conduzcan a una dinámica compleja donde las fluctuaciones crezcan y aparezcan patrones a distintas escalas espaciales y temporales. Este es un primer ingrediente básico en la física de la formación de patrones.

Los seres vivos son el ejemplo más evidente de sistemas fuera del equilibrio en los que un continuo intercambio de energía y materia es necesario para el mantenimiento de los mismos. Además, en general, los sistemas biológicos presentan una serie de estructuras desde el nivel celular al de organismos complejos pluricelulares. En estos últimos, su propia estructura de tejidos y órganos diferenciados son un prototipo de patrón biológico. La descripción física de cómo aparecen estas estructuras puede ser algo más complicada que la de los patrones no biológicos pero comparten ingredientes comunes básicos: interacción espacial entre componentes (en este caso, células) y no linealidad en la misma.

La primera y destacada incursión en la modelización de la formación de patrones en biología es debida a Alan Turing quien, en 1952, propuso un modelo sencillo de reacción-difusión basado en la físico-química de las componentes celulares (morfógeno) [1]. Desde aquel momento se han propuesto una gran variedad de mecanismos. Algunos de estos mecanismos son los modelos de activador-inhibidor, la difusión de morfógenos, el acoplamiento de osciladores genéticos... [2] La expresión matemática de estos mecanismos permite obtener, a partir de su solución, muchos de los patrones observados en la naturaleza.

Pese a que las ecuaciones de los modelos no suelen ser complicadas, la no linealidad de las interacciones dificulta mucho un tratamiento analítico adecuado. El problema se complica al considerar que los sistemas biológicos están en un entorno "ruidoso" por lo que es necesario emplear herramientas matemáticas y físicas propias de sistemas estocásticos. Todo esto hace que el desarrollo de técnicas computacionales sirva de puente entre los modelos matemáticos y los resultados experimentales. La simulación numérica es, por tanto, una herramienta imprescindible en este campo.

En este trabajo nos hemos centrado en un modelo de diferenciación celular mediante la proteína Notch. Este mecanismo ha sido fuertemente conservado en la evolución de organismos pluricelulares y es determinante en la formación de estructuras regulares en muchos tejidos. Actúa principalmente por tres procesos [3]: inhibición lateral, división asimétrica y formación de fronteras, los cuales se describen en la figura 1. Nuestro objetivo en este trabajo es combinar en un mismo estudio la diferenciación celular con el proceso de crecimiento de un tejido. En la sección 2 revisamos brevemente la biología de este mecanismo. A continuación vemos cómo se traducen las interacciones en un modelo matemático sencillo cuya solución abordaremos (numéricamente) en distintas geometrías.

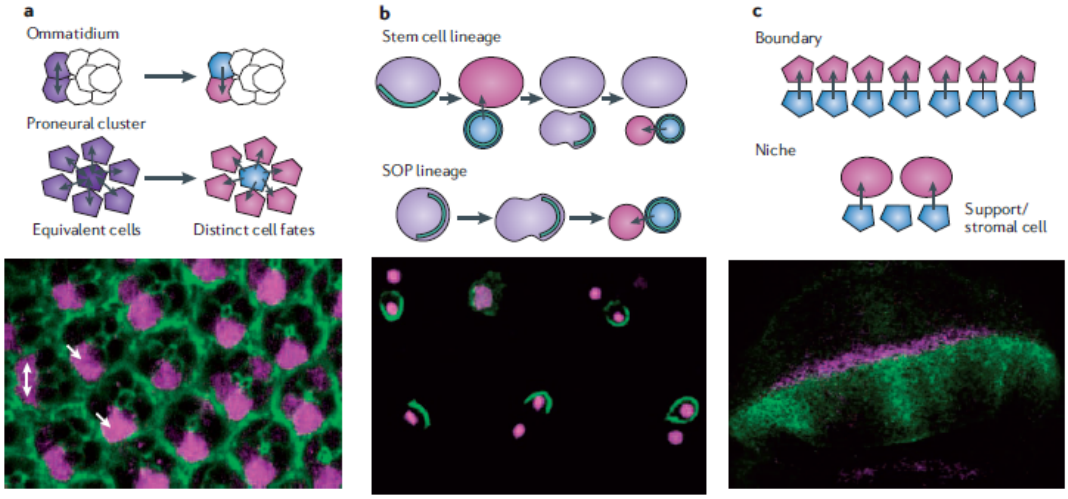


Figura 1: (**a**) **Inhibición lateral**: La señalización mediante Notch (flechas entre las células) permite que células equivalentes (en morado) se diferencien (en azul y rosa). La fotografía muestra un ojo de *Drosophila Melanogaster*, donde se han marcado en verde las membranas celulares y en rosa la actividad de Notch, la cual solo se detecta en una célula de cada omatidio. (**b**) **División asimétrica**: Durante la división, Notch (en verde) se distribuye de forma desigual entre las hijas. En la fotografía se observa la distribución de la proteína Numb, en verde, y los núcleos de células precursoras de los órganos sensores (SOP) de *Drosophila*. Se puede observar que Numb se encuentra solo en una de las células hijas. (**c**) **Fronteras**: La señalización de Notch forma una barrera que separa dos poblaciones. La fotografía corresponde a la formación del ala de *Drosophila*. Se aprecia una barrera entre células con alta presencia de Notch (en rosa) y otras con presencia de uno de sus ligandos (en verde). Imagen extraída de [3].

En la sección 3 planteamos un modelo sencillo de crecimiento tisular. Con este modelo se puede formar el sustrato adecuado para que se formen los patrones por el mecanismo de Notch. Finalmente los dos modelos se unen en un mismo modelo, que abarca el desarrollo de un tejido y la especialización de sus células.

2. El mecanismo Delta-Notch

En este trabajo se estudia el mecanismo de diferenciación celular conocido como inhibición lateral. En concreto, se estudia la interacción de dos proteínas denominadas Delta y Notch, que tienen homólogos en muchas especies animales.

El mecanismo de inhibición lateral consiste en la transmisión de una señal química entre células vecinas, la cual es responsable de inhibir la especialización de las células que rodean a otra célula que ya está predispuesta a especializarse. Estas células que se encuentran próximas a una célula con una especialización ya incipiente se especializarán de un modo diferente. El resultado es un tejido con células aisladas de un tipo rodeadas de otro tipo de células (patrón

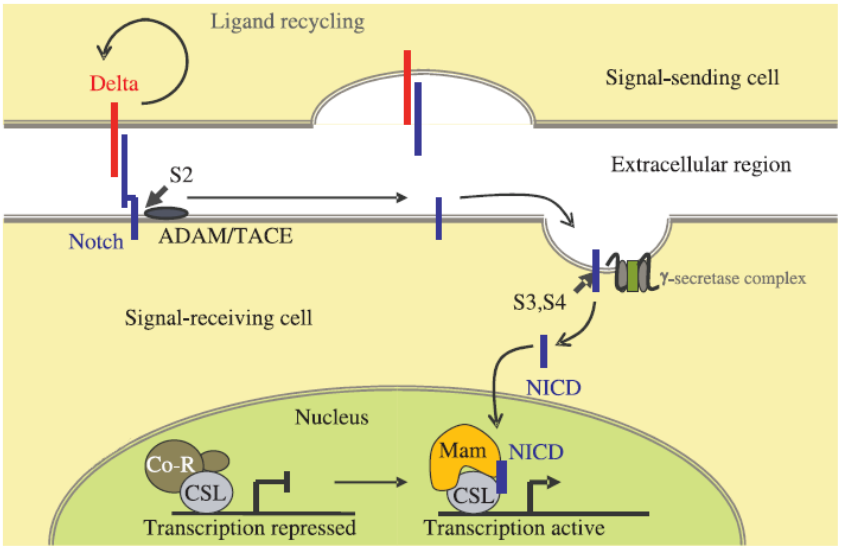


Figura 2: El diagrama muestra cómo actúa Notch y su interacción con Delta. Hay presentes varias especies que no se han descrito en el texto. Cuando Notch interacciona con Delta de otra célula, uno de sus dominios (NCID, *Notch Intracellular Domain*) entra en la célula y se dirige al núcleo para inhibir la transcripción de Delta y de los precursores de la especialización. Imagen extraída de [4].

salt and pepper). Se dice que las células aisladas se dirigen a un destino primario, el cual sería el de todas las células si no se produjese la inhibición lateral. El resto de células adquieren un destino secundario.

A grandes rasgos, el mecanismo Delta-Notch se puede comprender de la siguiente manera. Tanto Notch como sus ligandos son proteínas transmembrana. Existen varios ligandos de Notch, pero para el objetivo de este trabajo se considerará únicamente a Delta. Notch posee un dominio extracelular que puede interactuar con las proteínas Delta de células vecinas. Esta interacción conlleva una modificación de la estructura de Notch, la cual libera dentro de la célula su dominio intracelular, que tiene un efecto inhibidor sobre el gen que codifica Delta. En la figura 2 se presenta un esquema del mecanismo.

La interacción de Delta y Notch resulta en una red de retroalimentación que involucra a células vecinas. El efecto es el siguiente: una concentración alta de Delta activa la producción de Notch en las células vecinas. Una concentración alta de Notch desactiva la producción de Delta en la misma célula. Por tanto, las concentraciones de Delta y Notch en una misma célula serán complementarias: cuando una sea grande, la otra será pequeña. Una concentración elevada de Delta en una célula implica que las células vecinas adquieran una concentración elevada de Notch. Sin embargo, una concentración elevada de Notch no impide que las células vecinas produzcan una u otra proteína. El resultado es que algunas células tendrán alta concentración de Delta y estarán completamente rodeadas de células con alta concentración de Notch.

La presencia de Notch inhibe la expresión de los genes asociados al destino primario, actuando

como un factor de transcripción. Por tanto, las células que tienen una concentración elevada de Delta (y por tanto poca concentración de Notch) sí pueden alcanzar el destino primario, mientras que el resto de células, al ver impedida esta posibilidad, se dirigen al destino secundario. Por tanto, la apariencia final del tejido es de algunas células del tipo primario aisladas y rodeadas de células del tipo secundario.

La fisiología detrás de estos procesos es mucho más compleja [4]. No sólo se ha omitido la intervención de otras moléculas y de los mecanismos concretos en que actúan Delta y Notch, también se puede producir la diferenciación celular de otro modo, basado en la división celular asimétrica. En la segunda parte de este trabajo se pretende hacer una aproximación a este método.

2.1. Un modelo matemático del mecanismo Delta-Notch

Collier et ál. [5] proponen un modelo que ofrece resultados en consonancia con lo expresado en los párrafos anteriores. Este modelo se toma como la base de este trabajo.

Se toman como variables las concentraciones de las dos moléculas, Delta y Notch, en cada una de las células del tejido. Estas células se disponen en una red estática. El modelo se adapta fácilmente a diferentes ordenamientos de las células. Se toman los casos de dos células aisladas, de una red lineal de células y de una red bidimensional.

Se considera interacción únicamente entre células contiguas. Las concentraciones de Delta y Notch en cada célula vienen dadas por una ecuación diferencial para cada una de las variables. La variación de cada especie se deduce a partir de las siguientes suposiciones:

1. Cada célula solo interacciona con sus vecinas inmediatas.

2. La concentración de Notch es una función creciente de la concentración de Delta de las células vecinas.

3. La concentración de Delta es una función decreciente de la concentración de Notch en la misma célula

4. Tanto Delta como Notch están sujetas a degradación.

En la figura 3 se muestran las relaciones entre Delta y Notch que dan lugar a estas suposiciones. A partir de las mismas se puede escribir la forma que tendrán las ecuaciones diferenciales del modelo:

$$\begin{aligned} \dot{N}_i &= F(\bar{D}_i) - \gamma_N N_i \\ \dot{D}_i &= G(N_i) - \gamma_D D_i \end{aligned} \tag{1}$$

Las concentraciones de Notch y Delta de la célula i vienen representadas por N_i y D_i; el punto sobre una variable indica derivación temporal. $\bar{D}_i$ es el promedio de las concentraciones de Delta de las células vecinas a la célula i. γ_N y γ_D son las constantes de degradación de ambas proteínas. F es una función monótonamente creciente mientras que G es monótonamente decreciente.

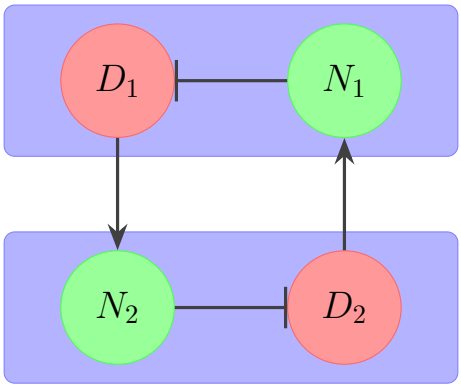


Figura 3: Esquema de la interacción entre las proteínas Delta y Notch de dos células diferentes. Las flechas indican activación, mientras que las barras corresponden a inhibición.

Las ecuaciones (1) contienen un término de producción y un término de degradación. Este último viene dado por una degradación exponencial. La degradación da cuenta de dos procesos: la destrucción de proteínas por procesos metabólicos y efectos asociados al aumento de volumen de la célula.

Para las funciones F y G se toman funciones de Hill. Este tipo de funciones sigmoidales son comúnmente utilizadas en la modelización de redes de regulación genética. Las funciones de Hill son de dos formas: de activación, la cual se aplica a F y de inhibición, la cual se aplica a G. Añadiendo estas funciones, las ecuaciones (1) toman la siguiente forma:

$$\begin{aligned} \frac{\mathrm{d}N_i}{\mathrm{d}t} &= \frac{\alpha_N \bar{D}_i^m}{\theta_D^m + \bar{D}_i^m} - \gamma_N N_i \\ \frac{\mathrm{d}D_i}{\mathrm{d}t} &= \frac{\alpha_D}{1 + \left(\frac{N_i}{\theta_N}\right)^h} - \gamma_D D_i \end{aligned} \tag{2}$$

En estas ecuaciones aparecen los parámetros α_D y α_N, que corresponden a la producción máxima de cada especie, y los parámetros θ_D y θ_N, que son los umbrales de concentración de cada especie para los cuales se inhibe o activa la producción de la otra especie a la mitad. Los exponentes m y h son los exponentes de las funciones de Hill. Cuanto mayores sean estos exponentes, más acusada será la curva sigmoidea de las funciones de Hill.

Para reducir el número de parámetros del modelo, se adimensionalizan las ecuaciones (2). Para ello, se definen nuevas variables adimensionales:

$$t = \frac{\tau}{\gamma_N} \qquad N_i = \theta_N n_i \qquad D_i = \theta_D d_i \tag{3}$$

Tras sustituir estas nuevas variables en las ecuaciones diferenciales del modelo, se definen también los siguientes parámetros:

$$\beta_d = \frac{\alpha_D}{\gamma_D \theta_D} \qquad \beta_n = \frac{\alpha_N}{\gamma_N \theta_N} \qquad \nu = \frac{\gamma_D}{\gamma_N} \tag{4}$$

De modo que β_d está relacionado con la producción de Delta, mientras que β_n marca la fuerza de la interacción entre células. Por su parte, el parámetro ν es el ratio entre los ritmos de degradación de Delta y Notch. El sistema tenía tres dimensiones: tiempo, concentración de Delta y concentración de Notch. Por tanto, se ha podido expresar en términos de otros tantos parámetros:

$$\begin{aligned} \frac{\mathrm{d}n_i}{\mathrm{d}\tau} &= \frac{\beta_n \bar{d_i}^m}{1+\bar{d_i}^m} - n_i \\ \frac{\mathrm{d}d_i}{\mathrm{d}\tau} &= \nu\left(\frac{\beta_d}{1+n_i^h} - d_i\right) \end{aligned} \tag{5}$$

Para comprender cómo estas ecuaciones dan lugar a la especialización celular en forma de patrones, se puede hacer un estudio cualitativo del sistema, a la luz de la figura 3. Considérense dos células vecinas. Comenzando con bajas concentraciones de Delta y Notch, la tendencia en ambas células será producir ambas proteínas. Pronto, Notch empezará a inhibir la producción de Delta en cada célula, lo cual a su vez implica que se reduzca (con cierto retraso) la producción de Notch en ambas células. Si ambas células mantienen siempre concentraciones similares, esto llevará a un equilibrio en el que las dos células tengan iguales concentraciones de las dos proteínas. Sin embargo, si hay una pequeña diferencia, la célula que tenga mayor concentración de Notch (por cualquier fluctuación sobre las concentraciones, sea en las condiciones iniciales o en la dinámica) inhibirá la producción de Delta en la propia célula, lo que conduce a producir Notch en la otra célula en menor medida y, en definitiva, que la otra célula aumente su producción de Delta. Puesto que esto activa la producción de Notch en la primera célula, el efecto se retroalimenta y amplifica, creando una asimetría entre ambas células: una tendrá una concentración elevada de Notch y la otra, de Delta; una célula se especializará conforme al destino primario y la otra, conforme al secundario.

Las ecuaciones presentadas son deterministas. Sin embargo, la naturaleza de los procesos involucrados (creación y degradación de proteínas) es estocástica. Por tanto, se debe añadir ruido a estas ecuaciones. Esto se podría hacer sumando una variable estocástica, simplemente, pero en esta situación el método correcto es añadir ruido multiplicativo, tal y como se demuestra en el anexo A. Las ecuaciones (5) contienen, cada una, un término de producción (las funciones de Hill), sea F_{xi}^+ y un término de degradación, sea F_{xi}^-. El ruido se añade de la siguiente forma:

$$\frac{dx_i}{dt} = F_{xi}^+ - F_{xi}^- + \sqrt{T\,\frac{F_{xi}^+ + |F_{xi}^-|}{2}}\,\eta_{xi}(t) \tag{6}$$

Siendo $\eta_{xi}(t)$ variables estocásticas de media cero y varianza $\langle \eta_{xi}(t)\eta_{x'j}(t')\rangle = 2\delta_{xx'}\delta_{ij}\delta(t-t')$, es decir, ruidos blancos gaussianos descorrelacionados. T es un parámetro que marca la amplitud del ruido. Aplicando esta expresión se reescriben las ecuaciones (5) de modo que se tenga en cuenta el ruido:

$$\begin{aligned} \frac{\mathrm{d}n_i}{\mathrm{d}\tau} &= \frac{\beta_n \bar{d_i}^m}{1+\bar{d_i}^m} - n_i + \sqrt{\frac{T}{2}\left(\frac{\beta_n \bar{d_i}^m}{1+\bar{d_i}^m} + n_i\right)}\,\eta_{ni}(t) \\ \frac{\mathrm{d}d_i}{\mathrm{d}\tau} &= \nu\left(\frac{\beta_d}{1+n_i^h} - d_i\right) + \sqrt{\frac{T}{2}\nu\left(\frac{\beta_d}{1+n_i^h} + d_i\right)}\,\eta_{di}(t) \end{aligned} \tag{7}$$

Estas ecuaciones definen el modelo Delta-Notch. Para estudiar el comportamiento del mismo, se integran numéricamente empleando los métodos expuestos en el apartado siguiente. Tal y como se han expresado, las ecuaciones se pueden aplicar a conjuntos de células con un orden arbitrario, siempre y cuando se calcule de manera adecuada el promedio de los niveles de Delta de las células vecinas. En concreto, se estudian los siguientes casos: dos células aisladas, células dispuestas en una red lineal y células en una red triangular.

2.2. Implementación del modelo Delta-Notch

Para integrar las ecuaciones diferenciales que dan la evolución de las concentraciones de Delta y Notch se utilizan dos algoritmos. Para las ecuaciones deterministas (5) se emplea el método de Runge-Kutta de cuarto orden. Por otro lado, para las ecuaciones estocásticas (7), se emplea el algoritmo de Heun [6], el cual permite implementar adecuadamente el ruido multiplicativo (véase anexo A; en el anexo B se ofrece una breve descripción de estos métodos).

Una parte fundamental del modelo es cómo se tienen en cuenta los primeros vecinos, para calcular el promedio $\bar{d}_i$. Esto depende de la topología de la red implementada. Se realizan simulaciones con tres topologías diferentes. En el caso de dos células, el cálculo de $\bar{d}_i$ es trivial: $\bar{d}_1 = d_2$ y $\bar{d}_2 = d_1$. Para una red lineal, este promedio también es sencillo de calcular:

$$\bar{d}_i = \frac{1}{2}(d_{i-1} + d_{i+1}) \tag{8}$$

Aparte de esta regla, se deben considerar las condiciones de contorno, es decir, cómo se calculan estos promedios para las células de los extremos. Se pueden tomar dos tipos de condiciones de contorno, abiertas o periódicas. Las condiciones abiertas implican simplemente que $d_0 = 0$ y $d_{n+1} = 0$, esto es, que no hay células más allá que los extremos (siendo n el número de células de la red). También se podría fijar un valor constante para d_0 y d_{n+1}. Sin embargo, las condiciones de contorno que se utilizan finalmente son periódicas:

$$d_0 = d_n \qquad d_{n+1} = d_1 \tag{9}$$

Que corresponde a considerar que la primera y la última célula están en contacto, como si las células formasen un anillo.

Otro tipo de redes requieren un método más sutil para determinar los primeros vecinos. De forma general, se define una matriz de conectividad T de tamaño $n \times n$ que recoge los contactos entre células. Si la célula i está en contacto con la célula j, el elemento de matriz T_{ij} será 1; si no lo están, será 0. Por tanto, esta matriz es simétrica. Además, se añade la condición $T_{ii} = 0$, pues una célula no toma su propio valor d_i para calcular el promedio $\bar{d}_i$.

La topología de la red que forman las células depende únicamente de cómo se defina la matriz de conectividad, T. El algoritmo se implementa de tal manera que tome la matriz T y calcule los promedios $\bar{d}_i$ de acuerdo a ella. En concreto, se implementa una red triangular, la cual corresponde a la disposición de las células en muchos tejidos. En el anexo C se expone la definición de la matriz de conectividad en el caso de una red triangular.

Una vez construida la matriz de conectividad, el promedio de Delta será:

$$\bar{d}_i = \frac{1}{z}\sum_{j=1}^{n} T_{ij} d_j \tag{10}$$

Donde z es el número de primeros vecinos. Con estos elementos es posible realizar simulaciones sobre el modelo Delta-Notch, el cual se implementa en un código en C++.

2.3. Resultados de las simulaciones

Las ecuaciones (5) (o (7) en su forma estocástica) definen el modelo Delta-Notch. Para caracterizar el mismo, se realiza una serie de simulaciones numéricas empleando las técnicas descritas en la sección anterior.

Dos células aisladas En el caso determinista, se integran las ecuaciones (5) empleando el algoritmo de Runge-Kutta de cuarto orden. Este algoritmo se aplican en primer lugar a dos células aisladas. Se toman los siguientes parámetros, extraídos de [7] (véase el anexo D para un listado completo de los parámetros):

$$\nu = 1 \qquad \beta_n = \beta_d = 50 \qquad m = h = 3 \tag{11}$$

Como condiciones iniciales, se toma siempre la ausencia de Notch ($n_1 = n_2 = 0$) y una concentración de Delta aleatoria en ambas células, generada mediante una distribución plana en el rango $(4 \times 10^{-4}, 6 \times 10^{-4}]$. El resultado de una de estas simulaciones se muestra en la figura 4. Para comprender el comportamiento del sistema conviene tener en cuenta que las ecuaciones (5) tienen tres puntos fijos cuando se aplican a dos células (se pueden encontrar análisis de la estabilidad del sistema en [5] y [8]). Uno de estos puntos fijos corresponde a un estado homogéneo en el cual ambas células tienen la misma concentración de Delta y Notch. Los otros dos puntos fijos son los estados heterogéneos en los cuales las células se diferencian, de modo que una tenga una alta concentración de Delta pero baja de Notch y la otra célula a la inversa.

La evolución de las simulaciones realizadas es la siguiente: las células se aproximan al estado homogéneo, produciendo Notch, pues la pequeña concentración de Delta inicial activa este proceso ($t < 1$ en la figura 4). Sin embargo, Notch empieza a inhibir a Delta, de modo que esta se degrada y la producción de Notch decae (hasta $t = 4$ en la figura 4). La concentración de Delta y Notch en ambas células se estabiliza en cierto punto, manteniéndose aproximadamente constante, con ciertas oscilaciones (hasta $t = 7$ en la figura 4). Si la concentración inicial de Delta es la misma en ambas células, el sistema se mantendrá en este estado, que corresponde al punto fijo homogéneo. Este punto fijo es inestable, lo que significa que una pequeña fluctuación separa al sistema del mismo. Por ello, la pequeña diferencia en las concentraciones iniciales de ambas células hace que el sistema se aparte del estado homogéneo y cada célula evolucione de forma diferente. En la figura 4, en la que la concentración inicial de Delta era menor en la célula 1, esta célula alcanza un estado en el que predomina Notch mientras que la segunda célula adquiere una alta concentración de Delta. A partir de $t = 10$ las concentraciones alcanzan un valor fijo tal que $n_1 = 50$ y $d_2 = 50$; estos valores corresponden a β_i, pues estos parámetros limitan asintóticamente la concentración de ambas proteínas. Puesto que la célula 1 tiene una

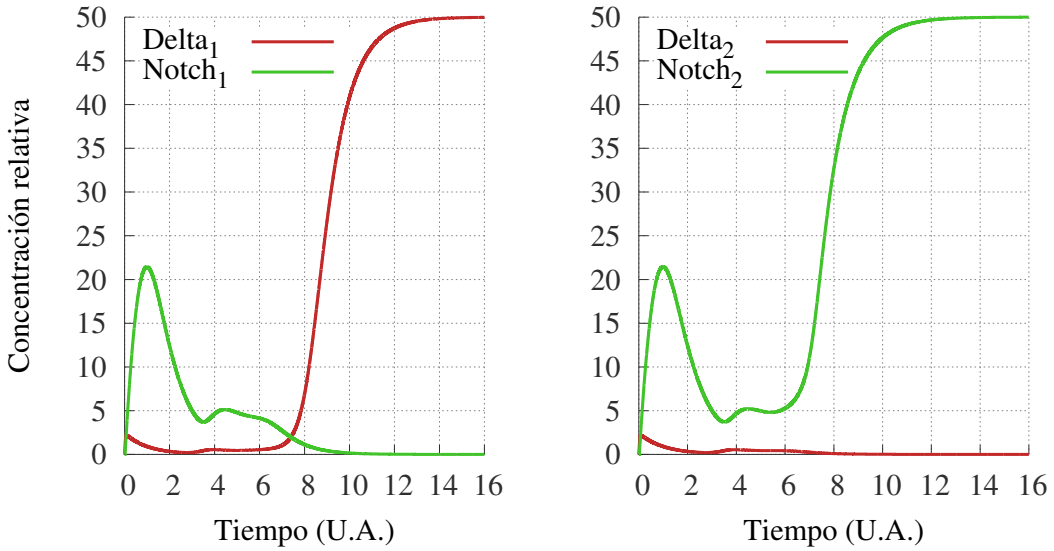


Figura 4: Variación con el tiempo de la concentración de Delta y Notch de dos células en contacto. Cada gráfica corresponde a una célula; en rojo se muestra la concentración de Delta y en verde la de Notch. Las unidades empleadas son arbitrarias.

alta concentración de Notch, se activará su función represora y la célula no alcanzará el destino primario sino el secundario. La célula 2, sin embargo, no verá inhibido el destino primario y este será al que se dirija.

Hay que remarcar que los valores seleccionados para los parámetros del modelo son tales que se den los tres puntos fijos. Otra selección de los parámetros puede implicar que solo exista el estado homogéneo, y que este sea estable, por ejemplo si las funciones de Hill no presentan cooperatividad, i. e. si $m = h = 1$ [7]. Este resultado no presenta interés para el fenómeno que se quiere estudiar, pues no conduciría a la diferenciación celular.

Red lineal de células De nuevo usando el algoritmo de Runge-Kutta de cuarto orden, se integran las ecuaciones (5) para un conjunto de células distribuidas en una red lineal. Se aplican las condiciones de contorno periódicas dadas por (9). Los parámetros del modelo son los mismos que en el caso de dos células, expuestos en las expresiones (11), con la excepción de los exponentes de Hill, que se toman $m = h = 4$ para que estas funciones sean pares, pues cuando las funciones de Hill son impares pueden aparecer valores negativos de la concentración de Delta y Notch. Como condiciones iniciales, se aplica una concentración de Notch nula a todas las células y una concentración aleatoria muy pequeña de Delta.

En la figura 5 se muestra una simulación en tres tiempos distintos. Las simulaciones avanzan siempre de una manera similar: en primer lugar, crece Notch de forma homogénea hasta llegar a cierto umbral en que deja de crecer (se pierde la activación de Notch debida a delta; primer fotograma de la figura). A partir de este punto, en función de pequeñas fluctuaciones en las concentraciones entre las células debidas a las condiciones iniciales, unas células tenderán a pro-

ducir Delta mientras que otras tenderán a producir Notch (segundo fotograma), produciéndose de esta manera la diferenciación celular.

Al cabo de un cierto tiempo se alcanza un estado estacionario en el cual todas las células han alcanzado el límite en la concentración de Delta o Notch (marcado, de nuevo, por β_i). En este estado, que corresponde al tercer fotograma de la figura 5, todas las células han alcanzado la especialización, sea primaria (células en rojo en la figura, con alta concentración de Delta) o secundaria (en verde, con alta concentración de Notch). Sin embargo, la interacción entre Delta y Notch impide que alcancen el destino primario dos células contiguas. No obstante, sí pueden darse dos células contiguas con especialización secundaria. Esto se considera como un defecto, puesto que rompe el patrón alternante que tiende a producirse en la red.

La diferenciación celular se empieza a dar en la red entre células cuyas concentraciones iniciales tengan mayor fluctuación y a partir de ahí se extiende a células vecinas. La diferenciación se propaga de modo similar a un frente de ondas. Si la red es lo suficientemente grande, es fácil que la diferenciación comience a la vez en varios puntos de la misma, por lo que esos frentes de diferenciación acabarán por encontrarse. Allí donde se encuentren, dependiendo de cuántas células hayan atravesado, se podrá formar un defecto. Comparando los fotogramas segundo y tercero de la figura 5 se puede comprobar que los defectos que se aprecian en el tercero coinciden con localizaciones en la red que han tenido una evolución más lenta, es decir, su nivel de Notch en el segundo fotograma es menor que el de las células próximas. Si los dos frentes de diferenciación se encuentran de forma que su propagación coincida con la periodicidad de la red (que corresponde a dos células), no darán lugar a un defecto.

Se aprecia, por tanto, cómo la formación de estos defectos depende fuertemente de las condiciones iniciales. Para que se dé la diferenciación, tiene que haber ciertas diferencias entre las concentraciones de las células, las cuales se han introducido por medio de aleatoriedad en las condiciones iniciales. Esto significa introducir ruido estático en la simulación. En realidad los sistemas biológicos se caracterizan por su naturaleza estocástica, por lo que un modelo realista debe incluir el ruido en su dinámica. Esto se hace mediante las ecuaciones (7), las cuales se integran con el algoritmo de Heun [6].

Las simulaciones con ruido son cualitativamente similares a las del caso determinista, pero se observan fluctuaciones en las concentraciones de Delta y Notch de todas las células durante toda la simulación. El estado estacionario es ahora un estado dinámico, en el que las variables fluctúan en torno a los valores de equilibrio. La amplitud de las fluctuaciones viene determinada por la amplitud del ruido estocástico, de modo que si este ruido es muy grande puede desestabilizar al sistema hasta el punto de impedirle llegar a un estado estacionario.

Una característica del sistema interesante es la relación entre el ruido estocástico y el número de defectos que aparecen en la red. Realizamos un estudio de esta relación para comprobar si el ruido ayuda a la formación del patrón o por el contrario la dificulta. Para este fin se define un parámetro de orden.

La idea de partida es que en un patrón perfecto, por cada célula en el destino primario debe haber otra en el destino secundario. Sin embargo, conforme aparecen defectos el número de células en el destino secundario será mayor que en el destino primario. Por tanto, el cociente entre ambas cantidades da cuenta del orden del sistema.

En primer lugar se debe definir cuándo una célula se ha especializado, para lo cual se tiene

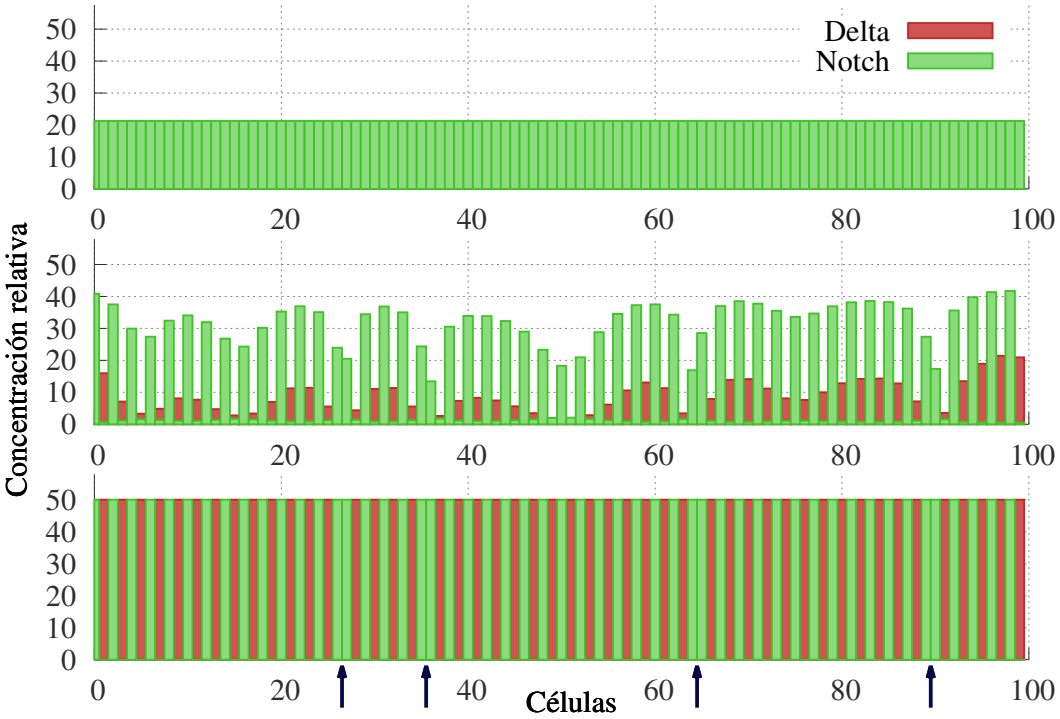


Figura 5: Concentración de las proteínas Delta y Notch en cada célula de una red lineal en tres instantes diferentes: la gráfica superior corresponde a $t = 1$, la de en medio a $t = 10$ y la inferior a $t = 20$; el tiempo es adimensional y sus unidades, arbitrarias. Las flechas marcan la posición de los defectos producidos. Se han integrado las ecuaciones (5) mediante el algoritmo de Runge-Kutta de cuarto orden.

en cuenta el límite asintótico dado por $\beta_i \equiv 50$. Se toma un umbral próximo: $d_i > 40$ para la especialización primaria y $n_i > 40$ para la secundaria. Se promedia el valor de d_i y n_i durante un cierto número de pasos para decidir si se ha especializado. A continuación se cuenta el número de células especializadas de cada tipo y se realiza el cociente:

$$\eta = \frac{N_d}{N_n} \tag{12}$$

Este parámetro de orden vale 1 en el caso perfectamente ordenado y $\frac{1}{2}$ si se da el número máximo de defectos posible (dos células en el destino secundario por cada célula en el primario). El caso en que se especialicen tres células contiguas por la vía secundaria es altamente inestable, puesto que la tendencia es especializarse por el destino primario tantas células como sea posible de modo que no haya dos contiguas en este destino.

El parámetro de orden (12) se mide para diferentes valores de la amplitud del ruido. Se toma como límite superior el valor $T = 0,6$ a partir del cual es muy complicado medir el parámetro de orden debido a que no se alcanza un estado estacionario. Las medidas se recogen en la figura 6. El parámetro de orden resulta ser monótonamente decreciente con la amplitud del ruido, es decir, el ruido empeora la calidad de los patrones. La dependencia es más acusada cerca del límite determinista, cuando el ruido tiende a cero. Por tanto, la formación óptima de los patrones se da cuando hay ausencia de ruido. No obstante, la dependencia es débil, siendo poca la diferencia en el número de defectos que aparecen a niveles altos o bajos de ruido. Esto se ve en la miniatura incluida en la figura 6, en la que se ha utilizado una escala para el parámetro de orden tal que se muestren todos los valores que puede tomar (entre 0 y 1).

Esta dependencia significa que el efecto global de las fluctuaciones propias del metabolismo de las células, que se modelizan mediante este ruido, es muy tenue. El resultado final, en cuanto a aspecto del patrón, no es muy diferente si estas fluctuaciones son mayores. El efecto que tiene el ruido se aprecia en la dinámica, pues las variaciones implican que haya más puntos de la red en los que se inicie la diferenciación. La dependencia leve con los procesos estocásticos da cuenta de la robustez matemática del modelo, el cual puede alcanzar su estado final adecuadamente pese a la fluctuación estocástica.

Red bidimensional triangular Aplicamos también las ecuaciones (7) al caso de una red triangular con condiciones de contorno periódicas. Los resultados de varias de estas simulaciones se observan en la figura 7.

Realizamos simulaciones en redes de tamaño 12×12 así como 24×24, con diferentes valores para la amplitud del ruido T. Los parámetros del modelo son los mismos que en el caso de la red lineal. El resultado general es análogo al de la red lineal: el sistema alcanza un estado homogéneo en el que todas las células tienen aproximadamente las mismas concentraciones y a continuación ciertas pequeñas diferencias en estas concentraciones se ven amplificadas. Finalmente, algunas células alcanzan elevadas concentraciones de Delta, lo cual implica que alcanzan el destino primario mientras que el resto se dirigen al destino secundario. Las primeras se encuentran siempre rodeadas por el segundo tipo, de forma que nunca haya en contacto dos células en el destino primario. En una red lineal el máximo ratio entre células en el destino primario y en el secundario era de 1 a 1, pero en el caso de la red bidimensional este ratio es menor, concretamente

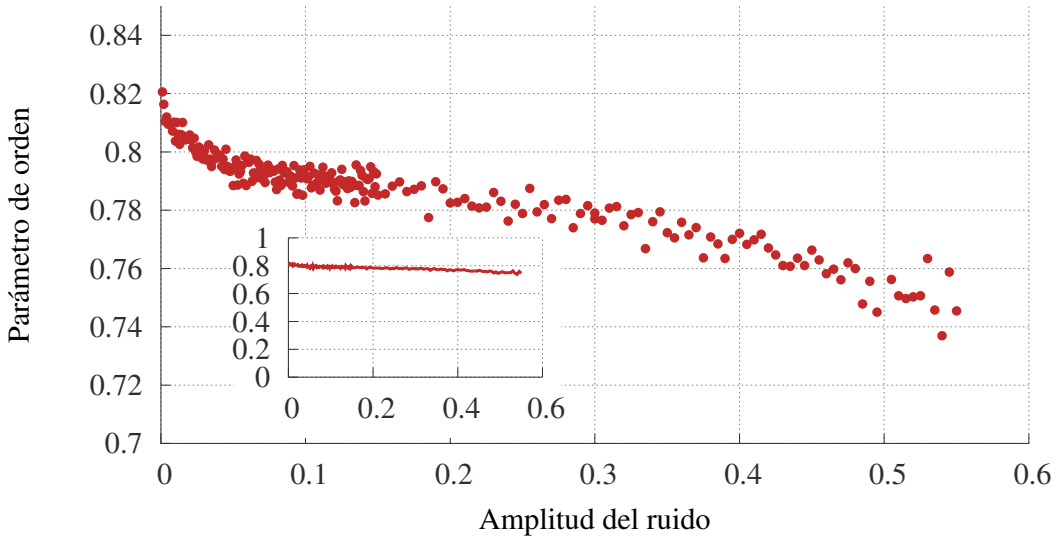


Figura 6: Valor del parámetro de orden (12) en función de la amplitud del ruido. Se ha utilizado una escala detallada para observar la forma de la curva obtenida; en la miniatura se observan los mismos datos con una escala absoluta (el parámetro de orden toma valores entre 0 y 1). Cada punto es el promedio de 200 medidas.

de una célula en el destino primario por dos en el secundario, como se puede apreciar en la figura 7F.

Con valores más elevados de ruido, el patrón formado contiene más defectos. La especialización se inicia en células individuales y se propaga de un modo similar a una onda. En este caso se propaga en dos dimensiones, lo que da lugar a paredes de dominio más complejas que en el caso de la red lineal. En las instantáneas A a C de la figura 7 se muestra la evolución de una simulación. En A se observan los puntos en que se inicia la especialización. Esta se propaga desde esos puntos, tendiendo a formar un patrón como el de F. Sin embargo, allí donde colisionan los frentes de onda se forman paredes de dominio que deforman el patrón.

Las figuras D y E corresponden a una red 12×12 con dos valores de la amplitud del ruido diferentes. El patrón obtenido es cualitativamente similar, sin embargo, si el ruido es muy alto las fluctuaciones pueden impedir la especialización efectiva, es decir, no se alcanza una concentración de Delta y Notch estable, lo cual se representa con colores tenues.

De forma similar a como hemos hecho para la red lineal, establecemos un parámetro de orden que dé cuenta de la regularidad del patrón formado. Observando el patrón perfecto de la figura 7F, se encuentra que los primeros vecinos de una célula especializada en el destino primario (las células rojas, con alta concentración de Delta) son todos células en el destino secundario. Sin embargo, exactamente la mitad de los segundos vecinos (es decir, 6 células) han alcanzado el destino primario. Si hay defectos, el número de segundos vecinos en el destino primario será siempre menor. Por tanto, un buen parámetro de orden puede consistir en contar el número de segundos vecinos que hayan alcanzado el destino primario. Este número se normaliza dividiendo

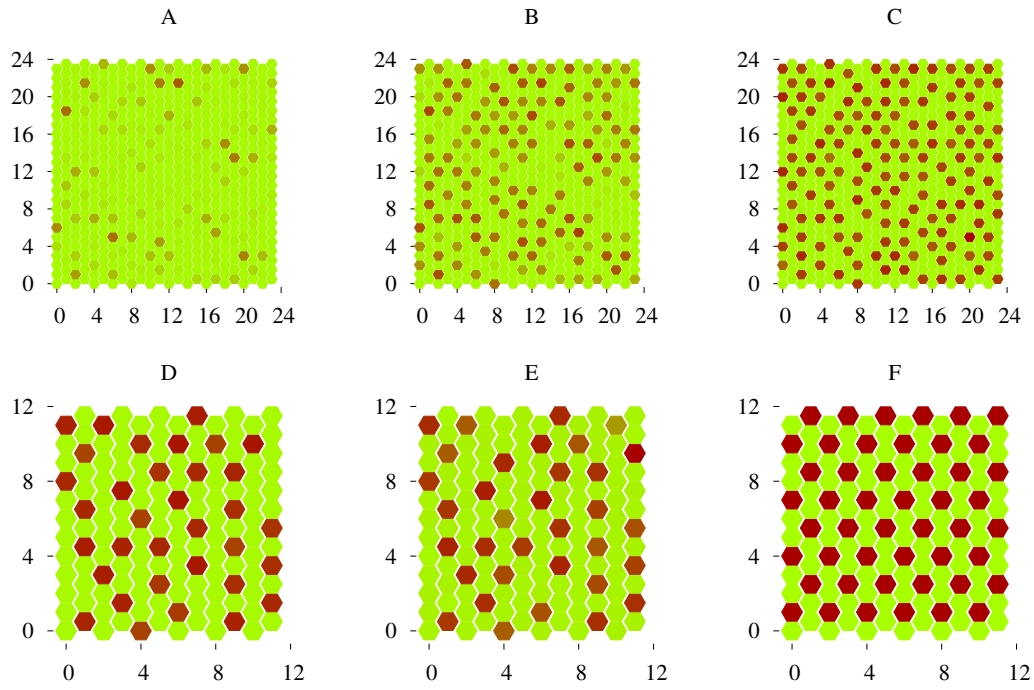


Figura 7: Diferentes representaciones de la red triangular de células. El color se corresponde con la concentración de Delta: rojo indica alta concentración, mientras que verde indica una concentración pequeña. A, B y C: red de 24×24 células en diferentes instantes de su evolución. La simulación se ha realizado con un ruido $T = 0,4$. D: red de 12×12 con un ruido $T = 0{,}3$. E: La misma red pero con ruido $T = 0{,}7$. F: Formación del patrón perfecto. Esta simulación se ha realizado sin ruido ($T = 0{,}0$) y con condiciones iniciales tales que favorezcan la formación de este patrón. En concreto, una de las filas de células especializadas comenzaban con una alta concentración de Delta. Para obtener este patrón se debe utilizar una red tal que su tamaño sea un número de veces entero la celda unidad. En el resto de simulaciones, las concentraciones tanto de Delta como de Notch se inicializan a 0.

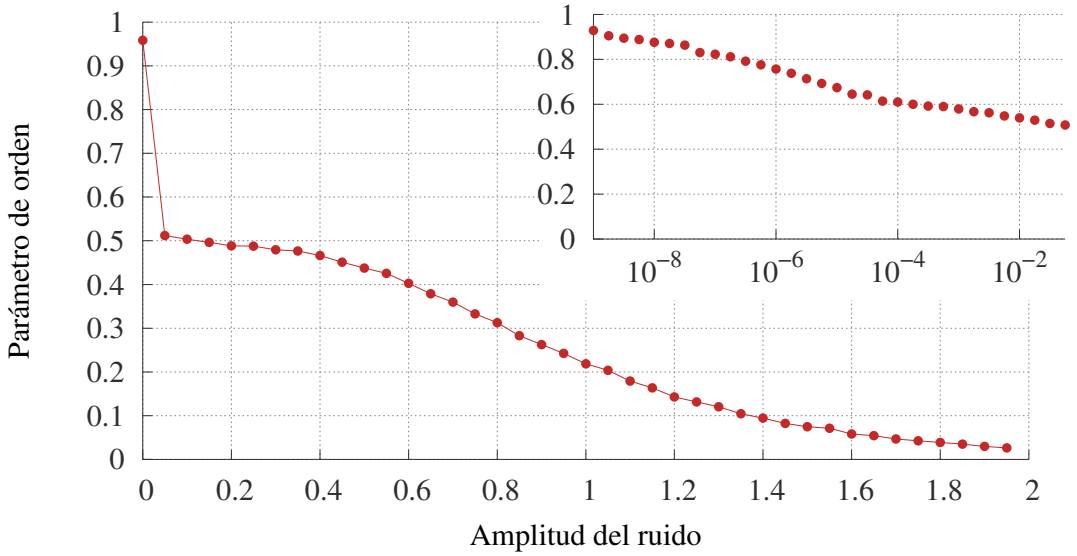


Figura 8: Representación del parámetro de orden definido para la red triangular, en función de la amplitud del ruido. En la miniatura se muestra, en escala semilogarítmica, la región de ruidos mucho menores que la unidad. Los errores no se incluyen puesto que son mucho menores que el tamaño de los puntos representados.

entre 6, de modo que en un patrón perfecto, el parámetro de orden será 1.

$$\eta_i = \begin{cases} \frac{1}{6}\sum_{j\in \text{s.v.}} \alpha_j & \text{si } d_i > 40 \\ 0 & \text{si } d_i \leqslant 40 \end{cases} \qquad \alpha_j = \begin{cases} 1 & \text{si } d_j > 40 \\ 0 & \text{si } d_j \leqslant 40 \end{cases} \tag{13}$$

Para extender la definición a toda la red, se suma el parámetro de orden de cada una de las células y se divide entre el número máximo de células en el destino primario, que es un tercio del número de células de la red. Con esta definición, el patrón perfecto de la figura 7F tiene un valor del parámetro de orden de 1. Conforme haya más defectos, este valor decrecerá, siendo 0 si no se especializa ninguna célula.

Se calcula este parámetro de orden para diferentes valores del ruido mediante el siguiente procedimiento: la red se inicializa y se deja evolucionar un cierto tiempo, de manera que se asegure que llegue a un estado de equilibrio. Entonces, se efectúa una medida del parámetro de orden. El proceso se repite 500 veces y se calcula el promedio de todas esas medidas. Se representa el resultado en la figura 8. Se obtiene una curva suave que muestra una transición entre ruidos bajos, para los cuales el parámetro es aproximadamente $0,5$ y ruidos altos, para los que el parámetro tiende a cero. Cabe destacar que si el ruido es nulo, el parámetro de orden es prácticamente 1, pero al aumentar el ruido cae rápidamente hasta $0,5$. Conforme aumenta el ruido, se da un mayor número de iniciadores de la especialización, de modo que aparecen más paredes de dominio que rompen la regularidad. Cuando el ruido es muy grande, el problema que surge es que la especialización no es estable y el número de células especializadas en cada instante se reduce.

La figura 8 muestra valores del ruido mayores que la figura 6. Esto es debido a que el parámetro de orden para la red triangular está bien definido si las células no se especializan (es cero) pero no ocurre así en el caso de la red lineal. Para comparar ambos parámetros de orden hay que tener en cuenta que su naturaleza es diferente.

El efecto del ruido, en los dos casos estudiados, es incrementar los defectos del patrón que se forma. El modelo propuesto es un modelo matemático que puede construir los patrones de forma determinista; añadir ruido tiene el sentido de tomar en consideración los efectos que se dan en un sistema real, tales como el tamaño reducido de la célula o la naturaleza estocástica de las reacciones entre las proteínas.

3. Un modelo de crecimiento de un tejido

Hasta ahora se ha considerado una red de células estática y sobre ella se han realizado las simulaciones, integrando las ecuaciones diferenciales del modelo Delta-Notch. Sin embargo, esto está muy alejado de la realidad de un tejido vivo, el cual tiene una alta variabilidad dinámica: las células se dividen y se dan fenómenos de migración celular y otros por los cuales su disposición espacial puede cambiar [9]. De hecho, mecanismos de diferenciación celular alternativos (o complementarios) a la inhibición lateral requieren la división de las células para tener lugar. Se trata, por ejemplo, de la división asimétrica, en la que cada hija adquiere un rol diferente. Se pretende desarrollar un modelo que añada estos elementos al modelo anterior. Por tanto, un primer objetivo es realizar un modelo que describa la división celular y el crecimiento de un tejido por la división de sus células.

Existen modelos de este tipo y software que los implementa, como por ejemplo *CellSys* [10] o *VCell* [11]. El modelo que aquí se desarrolla es más simple que los implementados en ese software, aunque se toman como referencia para su diseño.

3.1. Descripción del modelo de crecimiento

En primer lugar, se define cómo son las células. Se consideran células esféricas, caracterizadas por un radio R que da cuenta de su tamaño. El volumen de la célula, por tanto, será $\frac{4}{3}\pi R^3$. Las células pueden moverse por un plano y cada una viene dada por la posición de su centro de masas, en coordenadas x e y. Esto se puede visualizar como un cultivo celular creciendo sobre un cierto sustrato con el cual no tienen interacción más allá de restringir el movimiento de las células a su superficie.

Estas células se dividen y además deben tener algún movimiento para que las células nuevas puedan adaptarse al tejido. Por otra parte, en un tejido en crecimiento es de esperar que las células aumenten su tamaño. Esto además se relaciona con el proceso de división, pues en la división el volumen de la célula madre se debe repartir entre sus hijas. De esta manera, el modelo se divide en tres partes diferenciadas: crecimiento, división y movimiento.

El crecimiento se modela de una forma muy sencilla: las células crecen aumentando su radio de forma lineal. Puesto que el crecimiento puede estar afectado por muchos factores, se añade un cierto ruido aditivo. La ecuación para el tamaño de la célula i será, entonces [9]:

$$\dot{R}_i = G + \xi_i^G(t) \tag{14}$$

Donde G es el *factor de crecimiento*, el cual indica la rapidez a la que crecen las células, y $\xi_i^G(t)$ es una variable estocástica de media cero y varianza $\langle \xi_i^G(t)\xi_j^G(t')\rangle = \epsilon_G \delta(t-t')\delta_{ij}$. El resultado de integrar esta ecuación es un crecimiento lineal en el radio de las células con ciertas fluctuaciones cuya magnitud viene determinada por ϵ_G.

La división celular o mitosis viene marcada por el crecimiento de la célula. Se estipula como condición para que se produzca la división el que la célula alcance un cierto tamaño R_{umbral}. La división se produce de forma simétrica: cada célula recibe la mitad del volumen de la célula madre (en la simulación numérica no se divide *exactamente* cuando llega a R_{umbral}, debido a la discretización). En cuanto a la posición que ocupan las nuevas células, esta se determina de acuerdo a la interacción entre las células, la cual se detallará más adelante.

La última parte del modelo es la referente al movimiento de las células. Estas están sometidas al movimiento browniano, de modo que sus posiciones siguen la ecuación de Langevin; en concreto, en régimen sobreamortiguado, pues se desprecia el término de inercia debido a que los sistemas biológicos se hallan típicamente a bajo número de Reynolds [12].

$$\eta \dot{q}_i = -\boldsymbol{\nabla} V_i(\{\mathbf{r}_i\}) + \xi_i^L(t) \tag{15}$$

Aquí, q_i es cada una de las variables espaciales de cada célula, sea x_i o y_i, η es la viscosidad del medio, V_i es el potencial sobre la célula i creado por el resto de células y $\xi_i^L(t)$ es una variable estocástica, de nuevo con media cero y varianza $\langle \xi_i^L(t)\xi_j^L(t')\rangle = \epsilon_L \delta(t-t')\delta_{ij}$.

El potencial V_i marca la interacción entre las células. Este término es necesario puesto que en su ausencia, las células podrían superponerse: no se *verían* unas a otras. La forma de este potencial es fundamental para determinar cómo se organizarán espacialmente las células una vez se empiecen a dividir y aumentar en número.

Por una parte, las células deben mostrar una cierta repulsión cuando se aproximan: aunque las membranas celulares son flexibles y pueden deformarse, si una célula empuja a otra, esta responderá ejerciendo una cierta presión sobre la primera. Sin embargo, debe haber una cierta fuerza de adhesión que mantenga las células unidas, de modo que formen un tejido compacto y no se dispersen. Existen potenciales que modelan la interacción entre membranas celulares, como el potencial de Johnson-Kendall-Roberts [13], pero su complejidad, tanto a nivel físico como computacional, es excesiva para el efecto que se busca estudiar. Un candidato sería un potencial de Lennard-Jones pero las simulaciones que hemos realizado con este potencial no se asemejan al crecimiento de un tejido, sino a partículas libres con una cierta interacción que forma agregados.

Finalmente optamos por un potencial armónico, el cual tiene una serie de ventajas: tiene el aspecto cualitativo adecuado (repulsión para pequeñas distancias pero atracción si las células se separan) pero es computacionalmente sencillo y rápido tanto de implementar como de calcular. Además, un potencial más complejo se puede aproximar a un potencial armónico mediante un desarrollo en potencias. Este potencial debe tener su mínimo cuando las células estén en contacto.

No obstante, se debe añadir un ingrediente adicional a este potencial: las células deben interaccionar entre sí únicamente si se encuentran en contacto. Las células situadas a cierta

distancia no deberían *verse*. Esto se soluciona añadiendo una distancia R_{cutoff} a partir de la cual la interacción entre dos células se anula. Para determinar cuál debe ser esta distancia, se tienen las siguientes suposiciones: las células perfectamente ordenadas tienden a formar una red triangular y en tal caso una célula solo interacciona con sus primeros vecinos. Por tanto, la distancia R_{cutoff} tiene que ser tal que en una red triangular cada célula interaccione con sus primeros vecinos pero no con los segundos. Para asegurar que se forme el tejido y las células no se dispersen, tiene que estar próxima a la distancia a los segundos vecinos, pues en la dinámica de Langevin las células se pueden alejar un tanto de las posiciones de equilibrio. Esta distancia se extrae de la geometría de una red triangular cuando todas las células han alcanzado $R_i = R_{\text{umbral}}$ y se fija en $R_{\text{cutoff}} = 2,2R_{\text{umbral}}$.

En cuanto a la posición del mínimo del potencial o distancia de equilibrio entre dos células, debe ser la distancia a la cual ambas células están en contacto. Se define una distancia promedio $R_{\text{eq}} = 1{,}95R_{\text{umbral}}$, teniendo en cuenta que las células tienen un tamaño entre $0{,}8R_{\text{umbral}}$ y R_{umbral}, pues el radio de las células hijas es, aproximadamente, $2^{-3/2}R_{\text{umbral}}$, debido a que se reparte el volumen simétricamente.

El potencial será, finalmente:

$$V_i = \sum_{i \neq j} V_{ij} \quad \text{donde} \quad \begin{cases} V_{ij} = \frac{1}{2}k(d_{ij} - R_{\text{eq}})^2 & \text{si} \quad d_{ij} < R_{\text{cutoff}} \\ V_{ij} = 0 & \text{si} \quad d_{ij} > R_{\text{cutoff}} \end{cases} \tag{16}$$

Aquí, d_{ij} es la distancia entre las células i y j, es decir,

$$d_{ij} = |\mathbf{r}_i - \mathbf{r}_j| \tag{17}$$

Una vez definida la interacción entre las células, se puede especificar el mecanismo de división. Para determinar las posiciones de las nuevas células, se sitúan a la distancia de equilibrio de acuerdo con el potencial ($d_{ij} = R_{\text{eq}}$) y se varía el ángulo respecto al eje x que forma la dirección que las une, de forma que se minimice el potencial total al que están sometidas ambas células. Para ello, se utiliza el algoritmo de minimización por sección áurea [14].

Hay una especificación adicional a tener en cuenta para llevar a cabo la división celular. Si todas las células se dividen en cuanto alcanzan cierto tamaño, la estructura del tejido se deforma mucho cuando se produzca esa división. Para evitar esto, se impone la condición de que las células se dividan en función de si tienen células a su alrededor o no. Concretamente, una célula se divide únicamente si hay menos de 6 células a una distancia menor que R_{cutoff}. Como efecto de esta condición, el tejido tiende a crecer desde sus fronteras. Para evitar que las células crezcan de forma indefinida, también se limita el crecimiento cuando supera el radio umbral para la división.

Con esto se han determinado las tres partes que componen este modelo: el crecimiento, el movimiento y la división. Se implementa el modelo en C++, empleando el algoritmo de Runge-Kutta estocástico de segundo orden para integrar las ecuaciones de crecimiento y movimiento.

Parámetro de orden hexático Se espera que este modelo dé lugar a un tejido en el que las células se organicen en una estructura similar a la de la red triangular utilizada en la primera parte de este trabajo. Por tanto, para comprobar el buen funcionamiento del modelo resulta

conveniente cuantificar en qué medida se reproduce esta red. Esto se puede hacer mediante el parámetro hexático [15], el cual se define para cada célula j de la siguiente manera:

$$\psi_j = \frac{1}{n_j} \sum_k \exp(i6\theta_{jk}) \tag{18}$$

El número de primeros vecinos es n_j, definidos como aquellas células que estén a una distancia menor a R_{cutoff}. La suma se extiende a los primeros vecinos de la célula j. θ_{jk} es el ángulo que forman dos células entre sí, tomando el eje x en sentido positivo como referencia.

El promedio de este parámetro, que varía entre 0 y 1, para todas las células en cada instante da cuenta de lo ordenadas que están, en comparación con una red triangular.

3.2. Simulaciones del modelo de crecimiento

El resultado de una simulación numérica de este modelo se muestra en la figura 9. En los seis fotogramas que componen esta figura, las células se dividen formando un agregado compacto. La estructura que surge es la de una red triangular pero esta presenta algunos defectos. Se forman paredes de dominio allí donde las células no se ordenan perfectamente, las cuales se ven como líneas verdes. El crecimiento del tejido tiene lugar tanto en las fronteras del mismo como en los límites entre dominios, gracias a la condición de que células con seis vecinos no deben dividirse. Las células que se encuentran en el interior de uno de estos dominios ordenados están en un estado de alto equilibrio, minimizando el potencial, pero su división o un cierto desplazamiento supondría una perturbación considerable para el tejido.

Para estudiar la relación que hay entre el ruido ξ^L y la ordenación de las células, se calcula el parámetro hexático para diferentes valores de ξ^L. El método para ello consiste en inicializar el sistema, hacerlo evolucionar durante $0,08$ unidades de tiempo (un tiempo suficiente para que haya un número de células aceptable, que minimice los efectos asociados a los bordes del tejido) y a continuación medir, repitiendo el proceso 1000 veces para cada valor de ξ^L y promediando las medidas.

Las medidas se recogen en la figura 10, donde se observa que hay un máximo en torno a $\xi^L = 70$. El movimiento browniano de las células permite que se recoloquen adaptándose al potencial subyacente. Si es demasiado bajo, las células tardarán en adaptarse al potencial y la división será demasiado rápida para que el sistema encuentre el equilibrio. Por otro lado, un ruido demasiado elevado vencerá las fuerzas de contacto, sacando a las células de la posición de equilibrio. Un compromiso entre ambos efectos permite encontrar un nivel de ruido óptimo, correspondiendo con el máximo de la figura.

3.3. Uniendo las piezas

Una vez desarrollados, implementados y caracterizados ambos modelos, se procede a unirlos para formar un modelo único. El modelo que aúna a los otros dos describe una población de células que crecen, se dividen y se especializan.

Se deben tener algunas consideraciones para poder unir ambas implementaciones. En primer lugar, se debe definir qué son los primeros vecinos en un modelo continuo. Retomando la idea utilizada para el potencial, se considera que dos células son vecinas si están a menor distancia

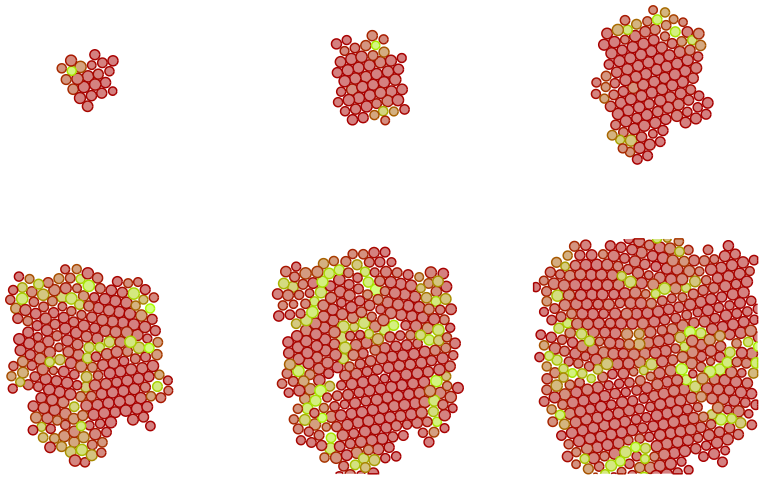


Figura 9: Seis fotogramas de una simulación: las células se representan mediante círculos. El color de cada célula se corresponde con el valor del parámetro hexático asociado: rojo significa que es próximo a 1 (muy ordenado) y verde significa que es próximo a 0 (muy desordenado). Los parámetros de la simulación son: $G = 30$, $\epsilon_G = 5 \times 10^7$, $\epsilon_L = 70$, $k = 3 \times 10^4$. Los tiempos que corresponden a cada fotograma, de izquierda a derecha y empezando por la fila de arriba son: $0,025$ unidades de tiempo; $0,040$ u. t.; $0,060$ u. t.; $0,073$ u. t.; $0,080$ u. t.; $0,090$ u. t.

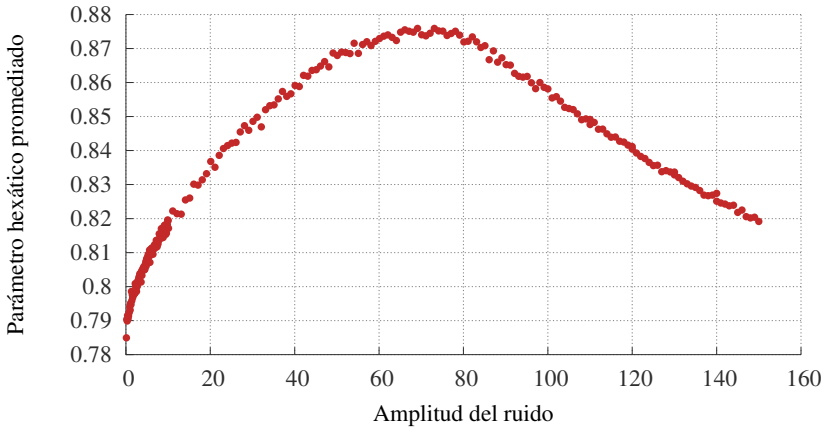


Figura 10: Medidas del parámetro hexático en función de la amplitud del ruido, ϵ_L. Cada punto corresponde al promedio de 1000 medidas. Se omiten los errores calculados pues son menores que el tamaño de los puntos representados.

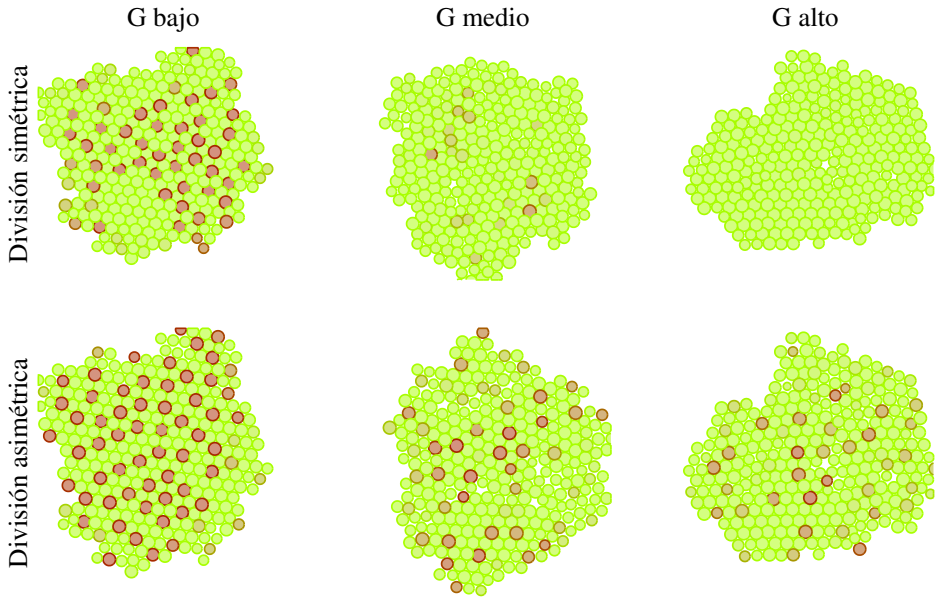


Figura 11: Simulaciones realizadas en seis casos distintos. Por un lado, se han simulado los casos en que el reparto de las proteínas Delta y Notch en la mitosis es simétrico o totalmente asimétrico. Por otro lado, se utilizan diferentes valores de G, constante que marca la escala de tiempo del crecimiento del tejido. Estos valores son tales que la división tenga una escala de tiempo menor, similar o mayor que el tiempo que tarda en producirse la diferenciación. Se han utilizado los valores $G = 5$, $G = 30$ y $G = 50$. Todas las simulaciones se han detenido cuando se han alcanzado 300 células. El color indica la concentración de Delta: rojo es alta concentración (destino primario) y verde, baja (destino secundario). Los tiempos que ha tomado cada simulación, de menor a mayor G (los tiempos son aproximadamente los mismos en los dos modos de división) son: $0,50$ unidades de tiempo, 0,08 u. t. y $0,05$ u. t.;

que R_{cutoff}. A partir de esta condición es inmediato construir una matriz de conectividad que, por cierto, será dinámica: cambia con el tiempo y debe ser calculada de nuevo en cada instante.

Por otra parte, se debe estipular cómo se reparten las proteínas Delta y Notch en la división celular. Hay dos posibilidades: *división simétrica*, de forma que cada célula reciba la mitad de cada especie (la concentración es constante tras la división) lo cual corresponde a considerar únicamente la inhibición lateral como mecanismo de especialización y *división asimétrica*, de forma que Delta y Notch no se repartan equitativamente. Esto puede dar lugar a un mecanismo de especialización adicional.

A fin de simplificar el modelo, no se tiene en cuenta la variación de la concentración de Delta o Notch con el tamaño de la célula.

En las figuras 11 y 12 se muestran resultados de estas simulaciones en diferentes casos. La diferencia entre las escalas temporales de crecimiento y especialización se controla modificando el factor de crecimiento, G. Si G es bajo, las células se van especializando conforme el tejido crece

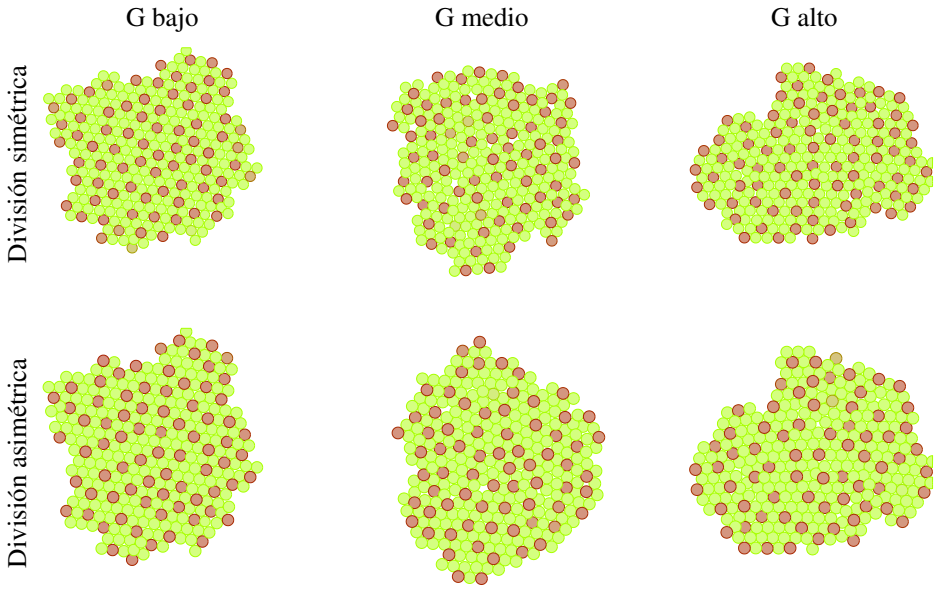


Figura 12: Se representan las mismas simulaciones que en la figura 11 pero dejándolas avanzar durante $0,6$ unidades de tiempo adicionales sin permitir división celular, de modo que Delta y Notch alcancen un estado estacionario.

y el patrón se va formando poco a poco, de modo que las nuevas células se adaptan a él una a una. La "onda de especialización" se propaga suavemente. Sin embargo, si G es grande, algunas células se especializan conforme crece pero al ser el crecimiento más rápido que la especialización, aparecen nuevas células que desplazan las que ya se habían especializado, deformando el patrón, lo cual perturba la propagación de la "onda de especialización".

Por otro lado, si la división es simétrica en cuanto a Delta y Notch, cuando una célula en el destino primario se divide, ambas hijas se inhiben mutuamente y se abandona la especialización, lo cual ralentiza mucho este proceso. Cuando la división es completamente asimétrica, si una célula se especializa se mantiene la especialización, así que los iniciadores se mantienen en el tiempo y la especialización ocurre más rápido (esto se pone de manifiesto en la figura 11).

En cuanto al patrón final, cualitativamente parece más regular si G es bajo y si la división es asimétrica, aunque sería necesario un estudio cuantitativo para obtener conclusiones precisas. Se requiere definir un parámetro de orden adecuado, cuya normalización es sutil puesto que el tejido es irregular y dinámico. Este estudio se deja como trabajo futuro.

En la figura 13 se muestra la dinámica en los casos de crecimiento rápido y lento. Si G es bajo, antes de cada división las células han podido especializarse, de modo que se forma el patrón en cada ciclo de división. Sin embargo, cuando G es alto la especialización toma varios de estos ciclos y el patrón no alcanza a formarse hasta que el tejido ha evolucionado lo suficiente como para estabilizarse.

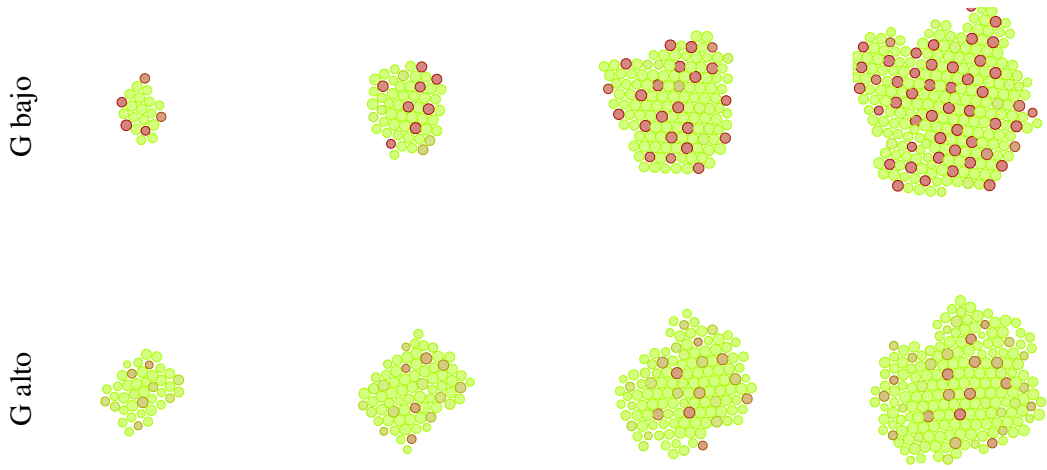


Figura 13: Diferentes fotogramas de dos simulaciones con división totalmente asimétrica y dos valores diferentes de G: $G = 5$ y $G = 50$. Los tiempos tomados son, para $G = 5$: $0,15$ unidades de tiempo, $0,25$ u. t.; $0,35$ u. t. y $0,45$ u. t. Para $G = 50$: $0,020$ u. t.; $0,026$ u. t.; $0,032$ u. t. y $0,040$ u. t.

4. Conclusiones

En este trabajo nos hemos centrado en dos modelos dedicados a la especialización de las células de un tejido formando patrones. El trabajo realizado se puede concretar en los siguientes puntos:

- Reproducción del modelo de Collier de Delta-Notch en un código en C++.
- Aplicación de este código a 3 sistemas: 2 células, red lineal y red triangular; aunque el código es flexible y admite otras geometrías.
- Estudio del comportamiento del modelo en tales sistemas en relación con el ruido, definiendo parámetros de orden adecuados.
- Desarrollo de un modelo de crecimiento tisular basado en primeros principios.
- Implementación de este modelo en un código en C++.
- Caracterización del modelo con ayuda del parámetro de orden hexático.
- Unión de ambos códigos y estudio cualitativo del modelo resultante.

A partir de la caracterización de los modelos hemos podido determinar que el ruido influye negativamente en la formación de un patrón regular en el modelo Delta-Notch y que el modelo de crecimiento tisular tiene un valor óptimo del ruido que actúa en el movimiento de las células,

de modo que estas se ordenan de acuerdo a una red triangular. En el modelo conjunto de ambos hemos observado cómo surge el patrón con diferentes relaciones entre las escalas temporales de los dos modelos y con diferentes modos de división en lo que concierne al reparto de Delta y Notch.

Sin embargo, estos modelos admiten mucho más desarrollo del que hemos podido llevar a cabo. Algunos puntos a desarrollar son:

- Se puede aumentar la complejidad del modelo de Delta-Notch, por ejemplo con el uso de modelos cinéticos, aplicando gradientes moduladores externos, teniendo en cuenta el área de contacto entre las células, considerando otros ligandos de Notch...
- Por su parte, el modelo de crecimiento tisular también admite mayor complejidad. Se pueden introducir potenciales de interacción más realistas y se puede tener en cuenta la muerte celular, por ejemplo.
- Ambos modelos contienen numerosos parámetros; sería interesante realizar un estudio cuantitativo detallado del espacio de parámetros.
- Los parámetros del modelo de crecimiento pueden adaptarse a la fisiología de organismos concretos.
- Este modelo sirve como base para otros circuitos genéticos que incluyan interacción entre células.
- Un mecanismo de especialización en el que interviene Notch es la formación de fronteras entre tejidos. Se puede estudiar el comportamiento del modelo conjunto cuando se incluyen paredes, o implementando condiciones de contorno periódicas.

Referencias

[1] Turing, A. M.: *The Chemical Basis of Morphogenesis.* Philosophical Transactions of the Royal Society of London, 237(641):37–72, 1952.

[2] Morelli, L. G., K. Uriu, S. Ares y A. C. Oates: *Computational Approaches to Developmental Patterning.* Science, 336:187–191, 2012.

[3] Bray, S. J.: *Notch signalling: a simple pathway becomes complex.* Nature, 7:678–689, 2006.

[4] Fiúza, U. M. y A. Martínez Arias: *Cell and molecular biology of Notch.* Journal of Endocrinology, 194:459–474, 2007.

[5] Collier, J. R., N. A. Monk, P. Maini *y cols.*: *Pattern formation by lateral inhibition with feedback: a mathematical model of delta-notch intercellular signalling.* J Theor Biol, páginas 429–446, 1996.

[6] Carrillo, O., M. Ibanes, J. Garcia-Ojalvo, J. Casademunt y J. Sancho: *Intrinsic noise-induced phase transitions: Beyond the noise interpretation.* Phys Rev E, 2003.

[7] Formosa-Jordan, P y D. Sprinzak: *Modeling Notch Signaling: A Practical Tutorial.* En *Notch Signaling - Methods and Protocols.* Springer Science+Business Media, 2014.

[8] Formosa-Jordan, P.: *Pattern formation through lateral inhibition mediated by Notch signaling.* Tesis de Doctorado, Universitat de Barcelona, 2013.

[9] Muñoz-García, Javier y Saúl Ares: *Formation and maintenance of nitrogen-fixing cell patterns in filamentous cyanobacteria.* Proceedings of the National Academy of Sciences, 113(22):6218–6223, 2016. `http://www.pnas.org/content/113/22/6218.abstract`.

[10] Hoehme, S. y D. Drasdo: *A cell-based simulation software for multi-cellular systems.* Bioinformatics, 26(20):2641–2642, 2010.

[11] Center for Cell Analysis and Modeling: *VCell - The Virtual Cell.* `http://www.vcell.org/`, 2016. accedido el 8 de junio de 2016.

[12] Purcell, E.M.: *Life at low Reynolds number.* American Journal of Physics, 45(1), 1977.

[13] Chu, Y. S., S. Dufour, J.P. Thiery, E. Perez y F. Pincet: *Johnson-Kendall-Roberts Theory Applied to Living Cells.* Phys Rev Lett, 94(028102), 2005.

[14] Press, W. H., S. A. Teukolsky, W. T. Vetterling y B. P. Flannery: *Numerical Recipes.* Cambridge University Press, tercera edición, 2007.

[15] Hedges, L. O.: *Structural order parameters (the easy way).* `http://lesterhedges.net/miscellanea/structural/`. accedido el 9 de junio de 2016.

[16] Sprinzak, D, A. Lakhanpal, L. LeBon, J. Garcia-Ojalvo y M. B. Elowitz: *Mutual Inactivation of Notch Receptors and Ligands Facilitates Developmental Patterning.* PLoS Comput Biol, 7(6), 2011.

[17] Kloeden, P. E. y E. Platen: *Numerical Solution of Stochastic Differential Equations.* Springer-Verlag Berlin Heidelberg, 1992.

[18] García-Ojalvo, J.: *Physical approaches to the dynamics of genetic circuits: a tutorial.* Contemporary Physics, 52:439–464, 2011.

[19] Kondo, S. y T. Miura: *Reaction-Diffusion Model as a Framework for Understanding Pattern Formation.* Science, 329:1616–1620, 2010.

[20] Gillespie, D. T.: *The chemical Langevin equation.* Journal of Chemical Physics, 113:297–306, 2000.

[21] Bialek, W.: *Biophysics: searching for principles.* Princeton University Press, 2012.

A. Ruido en dinámica celular (cinética química)

Las proteínas dentro de la célula se crean y se degradan. La dinámica de estos procesos se puede representar por la ecuación

$$\frac{\mathrm{d}N}{\mathrm{d}t} = s - kN(t) \tag{19}$$

donde s es la tasa de formación y k la de degradación. Esta ecuación puramente determinista es válida en el caso de N suficientemente grande en el que se puedan despreciar las fluctuaciones.

Esto no es el caso del interior de una célula donde el valor absoluto de algunas proteínas (como los factores de transcripción) son del orden de decenas o centenas de partículas. En este caso $N(t)$ es una variable estocástica y respondería a la ecuación (de Langevin) [20]:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = s - kN(t) + \xi(t) = a(t) + \xi(t) \tag{20}$$

La pregunta que se plantea es ¿cuáles son las propiedades de este término de ruido $\xi(t)$?

Una forma relativamente sencilla de obtener el término de fluctuación es partir de la "ecuación maestra" para este proceso. Sea $P(N,t)$ la probabilidad de tener N partículas en el tiempo t. Su evolución es:

$$\frac{\partial P(N,t)}{\partial t} = -sP(N,t) + sP(N+1,t) - kNP(N) + k(N+1)P(N) \tag{21}$$

La solución en estado estacionario de esta ecuación,

$$\frac{\partial P(N,t)}{\partial t} = 0 \tag{22}$$

corresponde a una distribución de Poisson [21]:

$$P(N) = e^{-M}\frac{M^N}{N!} \quad \text{con} \quad M = \frac{s}{k} \tag{23}$$

de donde se puede obtener:

$$\langle N \rangle = \sum_{N=0}^{\infty} NP(N) = M = \frac{s}{k} \tag{24}$$

Que es la solución, también, de la ecuación 19.

A partir de la ecuación maestra 21, suponiendo que N no es demasiado pequeña, se puede desarrollar:

$$P(N \pm 1, t) = P(N,t) \pm \frac{\partial P}{\partial N} + \frac{1}{2}\frac{\partial^2 P}{\partial N^2} \tag{25}$$

Sustituyendo en 21 y tras un cálculo un poco tedioso se obtiene

$$\frac{\partial P}{\partial t} = -\frac{\partial}{\partial N}(s - kN)P(N,t) + \frac{\partial^2}{\partial N^2}\left(\frac{1}{2}(s + kN)P(N,t)\right) \tag{26}$$

Esta es una ecuación de Fokker-Planck donde el primer término es una corriente de deriva (*drift*) que corresponde al término determinista $a(t)$ de la ecuación (20).

El segundo término es el de difusión, que da la amplitud del término estocástico $\xi(t)$:

$$\langle \xi(t) \rangle = 0 \quad \langle \xi(t).\xi(t') \rangle = \sqrt{\frac{(s+kN)}{2}} \delta(t-t') \tag{27}$$

En el caso, como el que se usa en este trabajo, de que las variables sean las concentraciones, si se define:

$$N = cV \quad \tilde{s} = \frac{s}{V} \quad \tilde{k} = k \tag{28}$$

Siendo V el volumen de la célula. La ecuación (20) queda:

$$\frac{\mathrm{d}c}{\mathrm{d}t} = \tilde{s} - \tilde{k}c + \tilde{\xi}(t) \tag{29}$$

donde $\langle \tilde{\xi}(t) \rangle = 0$ y $\langle \tilde{\xi}(t)\tilde{\xi}(t') \rangle = \frac{\tilde{s}+\tilde{k}c}{2V}$.

Identificando $\tilde{s}$ con la función de Hill de producción y $\tilde{k}$ con el coeficiente de degradación, se obtienen las expresiones (7) con $T = \frac{1}{V}$.

B. Métodos numéricos

En este anexo se exponen los algoritmos empleados para resolver numéricamente las ecuaciones diferenciales de los modelos estudiados.

B.1. Algoritmo de Runge-Kutta de cuarto orden

Dada una ecuación diferencial de la forma:

$$\dot{x} = f(x) \tag{30}$$

y dado el punto x_n, el siguiente punto se calcula empleando las siguientes expresiones:

$$\begin{aligned} k_1 &= f(x_n)\Delta t \\ k_2 &= f(x_n + \frac{1}{2}k_1)\Delta t \\ k_3 &= f(x_n + \frac{1}{2}k_2)\Delta t \\ k_4 &= f(x_n + k_3)\Delta t \\ x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{31}$$

Este algoritmo se ha utilizado para integrar las ecuaciones del modelo de Collier en el caso determinista.

B.2. Algoritmo de Heun

Se pretende integrar una ecuación diferencia de la siguiente forma:

$$\dot{x} = F(x) + G(x)\xi(t) \tag{32}$$

Donde $\xi(t)$ es una función estocástica con las propiedades siguientes:

$$\langle \xi(t) \rangle = 0 \qquad \langle \xi(t)\xi(t') \rangle = 2C\delta(t - t') \tag{33}$$

La componente estocástica se integra de la siguiente manera:

$$\Delta W = \int_t^{t+h} \mathrm{d}s\, \xi(t) = \sqrt{2C\Delta t} z \tag{34}$$

Siendo z un número aleatorio con distribución gaussiana. Estos números se calculan mediante el algoritmo de Box-Muller.

El valor de x en el siguiente paso se calcula en dos pasos; en primer lugar se calcula el denominado *predictor*:

$$\bar{x} = F(x)\Delta t + G(x)\Delta W \tag{35}$$

El valor definitivo viene dado por el *corrector*:

$$x(t + \Delta t) = \frac{1}{2}[F(x) + F(\bar{x})]\Delta t + \frac{1}{2}[G(x) + G(\bar{x})]\Delta W \tag{36}$$

B.3. Algoritmo de Box-Muller

Este sencillo algoritmo permite obtener números aleatorios con distribución gaussiana, de media cero y varianza la unidad. Se parte de un generador aleatorio con distribución plana: a partir de dos números obtenidos mediante este generador (d_1 y d_2) se obtienen dos números con distribución gaussiana:

$$\begin{aligned} z_1 &= \sqrt{-2\log d_1}\cos(2\pi d_2) \\ z_2 &= \sqrt{-2\log d_1}\sin(2\pi d_2) \end{aligned} \tag{37}$$

B.4. Algoritmo de Runge-Kutta de segundo orden estocástico

Este algoritmo se aplica a una ecuación como la siguiente:

$$\dot{x} = f(x) + \xi(t) \tag{38}$$

La integración se realiza en tres pasos:

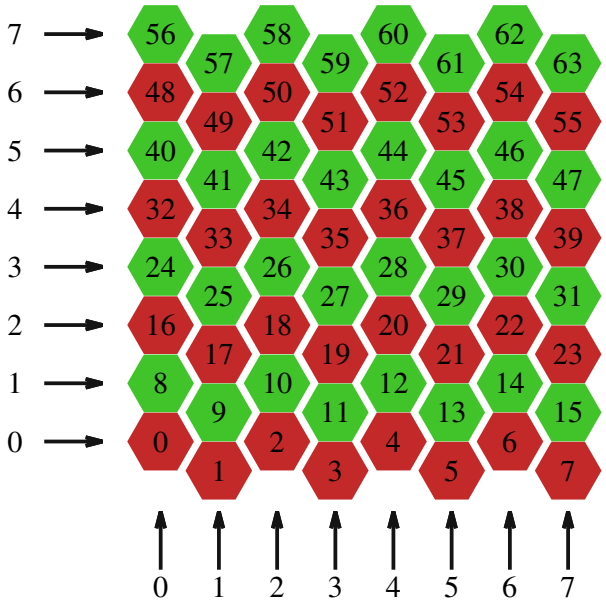


Figura 14: El diagrama indica cómo se han numerado las filas y columnas de la red triangular. Las células de cada fila se han representado en un mismo color. Dentro de cada célula se muestra el número que le corresponde.

$$\begin{aligned} g_1 &= f(x_0 + \sqrt{2C\Delta t}z) \\ g_2 &= f(x_0 + \Delta t g_1) \\ x &= x_0 + \frac{\Delta t}{2}(g_1 + g_2) \end{aligned} \tag{39}$$

donde z es de nuevo un número generado mediante el algoritmo de Box-Muller.

C. Matriz de conectividad para una red triangular

Cada célula de la red triangular se numera mediante un índice, que toma valores de acuerdo a la figura 14. También se numeran las filas y columnas, por lo que también se puede hacer referencia a una célula mediante una pareja de índices indicando en qué fila y en qué columna está. De esta forma, la matriz de conectividad se define como sigue:

$$\begin{aligned} & T_{i,i+1} = T_{i,i-1} = T_{i,i+N_c} = T_{i,i-N_c} = 1 \\ \text{Si } i \text{ está en columna par} \quad & T_{i,i+N_c-1} = T_{i,i+N_c+1} = 1 \\ \text{Si } i \text{ está en columna impar} \quad & T_{i,i-N_c-1} = T_{i,i-N_c+1} = 1 \end{aligned} \tag{40}$$

Donde N_c es el número de columnas que hay en la red. El resto de elementos T_{ij} son cero.

Además de esto, se aplican de nuevo condiciones de contorno periódicas. En este caso, puesto que se trata de una red bidimensional, las condiciones de contorno periódicas implican que los elementos de cada extremo sean contiguos a los del extremo opuesto. Esto es análogo a plegar la red en forma de toro. El tratamiento de estas condiciones de contorno es un tanto sutil, siendo diferente en función del tamaño de la red en cada coordenada. Todas las simulaciones se realizan con redes tales que el número de filas y columnas sean iguales y pares. Teniendo esto en cuenta, las condiciones de contorno se definen en diferentes casos: para las células de los bordes y para las células de las esquinas. Para las primeras se define:

$$\begin{array}{ll}
\text{Si } i \text{ está en el borde izquierdo o derecho:} & T_{i,i+N_c} = T_{i,i-Nc} = T_{i,i+N_c\pm 1} = 1 \\
\text{Si } i \text{ está en el borde inferior y columna impar:} & T_{i,i-N_c+n} = T_{i,i-N_c+n\pm 1} = 1 \\
\text{Si } i \text{ está en el borde superior y columna par:} & T_{i,i+N_c-n} = T_{i,i+N_c-n\pm 1} = 1 \\
\text{Si } i \text{ está en el borde inferior y columna par:} & T_{i,i-N_c+n} = 1 \\
\text{Si } i \text{ está en el borde superior y columna impar:} & T_{i,i+N_c-n} = 1
\end{array} \tag{41}$$

El resto de sus vecinos se obtienen mediante las expresiones anteriores. Las células de las esquinas presentan casos especiales que deben tratarse manualmente. Se comprueba que efectivamente se calcula la matriz correctamente mediante un código que muestra los primeros vecinos de cada una de las células, de forma secuencial.

D. Valores de los parámetros de los modelos

En la tabla siguiente se recogen los valores de los parámetros que se han utilizado en las simulaciones que se han expuesto a lo largo de la memoria. Para los parámetros que se han variado en algún momento, se muestra el rango de variación. En el caso de ϵ_L se añade el valor óptimo extraído del análisis realizado.

El parámetro η se ha fijado en 1, lo cual es equivalente a adimensionalizar el tiempo. Esto permite relacionar las escalas de tiempo del modelo Delta-Notch, dado por las ecuaciones (7), y el modelo de crecimiento tisular, dado por (14) y (15).

El paso temporal, Δt, es el mismo en todas las simulaciones y es global para ambos modelos.

Modelo Delta-Notch

Parámetro	Valor (o rango)
β_n	50
β_d	50
ν	1
m	3-4
h	3-4
T	0,0 a 2,0
Δt	1×10^{-5}

Modelo de crecimiento tisular

Parámetro	Valor (o rango)
η	1
k	3×10^4
ϵ_L	0,0 a 150,0; (70,0)
G	5 a 50
ϵ_G	5×10^7
R_{umbral}	1
Δt	1×10^{-5}

E. Códigos

A lo largo del trabajo se han desarrollado una serie de códigos en C++ que implementan los dos modelos tratados. En su mayor parte, son versiones de un mismo código para cada modelo, adaptados a diferentes contextos. A continuación se listan algunos de los códigos más representativos y se reproducen una versión final de cada modelo.

- `dos_celulas_delta_notch`
  Integra las ecuaciones (5) para dos células mediante Runge-Kutta de cuarto orden.

- `red_lineal_delta_notch`
  Integra las ecuaciones (5) en una red lineal de células mediante Runge-Kutta de cuarto orden.

- `red_lineal_delta_notch_estocastico`
  Integra las ecuaciones (7) en una red lineal de células mediante Heun.

- `red_triangular_delta_notch`
  Integra las ecuaciones (7) en una red triangular de células mediante Heun.

- `GLM_lennard_jones`
  Implementación del modelo de crecimiento tisular con sus tres partes: Crecimiento (G, *growth*), movimiento (L, Langevin) y mitosis (M). Se aplica un potencial de Lennard-Jones entre las células.

- `GLM_armonico`
  Implementación GLM con un potencial armónico entre las células.

- `GLM_delta_notch`
  Fusión de los modelos Delta-Notch y de crecimiento.

Los códigos *GLM* están compuestos de varios ficheros, pues se han elaborado una serie de librerías auxiliares para conseguir una mejor organización del código y facilitar el versionado. Estas librerías son las siguientes:

- `cell_class`
  Contiene la definición de la clase célula.

- `random_numbers`
  Recoge los generadores de números aleatorios.

- `algoritmos_para_minimizar`
  Incluye funciones que permiten obtener eficientemente el mínimo de una función.

- `GLM_graficos`
  Se encarga de la representación gráfica de resultados.

- `modelo_delta_notch`
  Contiene las funciones correspondientes al modelo Delta-Notch, aplicadas al modelo de crecimiento.

A continuación se reproducen los códigos `red_triangular_delta_notch` y `GLM_delta_notch` (con sus correspondientes librerías).

E.1. `red_triangular_delta_notch`

```
,
//LIBRERIAS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//CONSTANTES
#define nu 1
#define beta_d 50
#define beta_r 50
#define h 4 //h y m son los coeficientes (exponentes) de Hill. Son exponentes que aparecen en la funcion de Hill.
#define m 4
#define t_step 0.00001    //Paso temporal
#define N_pasos 20000   //Numero de pasos de Heun que se realizan
#define tamanno_red 144
#define Nc 12   //Numero de columnas de la red
#define Nf 12   //Numero de filas de la red
#define n_primeros_vecinos 6    //es el numero de primeros vecinos que tiene cada celula
//Constantes estocasticas
#define Temperatura 0.0 //Marca la amplitud del ruido
double epsilon; //Se define como variable y no como constante para calcularlo una unica vez pero usarlo en todo el programa

#define pi 3.141592
#define NormRANu (2.3283063671E-10F)            //Parisi-Rapuano
extern float ParisiRapuano(void);   //Genera numeros aleatorios con distribucion plana en (0,1]
extern void ini_ran(int seed);  //Inicializa el generador Parisi-Rapuano
void BoxMoller (double d1, double d2, double *z1, double *z2);       //d1 y d2 deben ser dos numeros con probabilidad plana; z1 y z2 son dos numeros devueltos con probabilidad gaussiana.
unsigned int irr[256];
unsigned int ir1;
unsigned char ind_ran, ig1, ig2, ig3;

void pasoHeun(double *delta, double *repressor); //Hace un paso de Heun
double frepressor(double repressor_i, double delta_j); //ecuacion diferencial para la actividad del repressor
double fdelta(double repressor_i, double delta_i);//ecuacion diferencial para la actividad de delta
```

```
35 double grepressor(double repressor_i, double delta_j); //Parte que multiplica al
       ruido en la ecuacion del repressor
36 double gdelta(double repressor_i, double delta_i); //Parte que multiplica al
      ruido en la ecuacion de delta
37 double valor_absoluto (double x);   //Implementacion de valor absoluto para
      doubles
38 void construir_matriz_conectividad(bool M[tamanno_red][tamanno_red]);//Modificar
       esta matriz permite utilizar otras geometrias
39 double promediar_delta_vecinos (double *delta, int i_cell);
40
41 bool M_conectividad[tamanno_red][tamanno_red];
42
43 extern void ini_ran(int seed)
44 {
45     int ini, factor, sum, i;
46     srand(seed);
47     ini = seed;
48     factor = 67397;
49     sum = 7364893;
50     for(i=0; i<256; i++)
51     {
52         ini = (ini*factor) + sum;
53         irr[i] = ini;
54     }
55     ind_ran = ig1 = ig2 = ig3 = 0;
56 }
57
58 float ParisiRapuano(void)
59 {
60     float r;
61     ig1 = ind_ran-24;
62     ig2 = ind_ran-55;
63     ig3 = ind_ran-61;
64     irr[ind_ran] = irr[ig1]+irr[ig2];
65     ir1 = (irr[ind_ran]^irr[ig3]);
66     ind_ran++;
67     r = ir1*NormRANu;
68     return r;
69 }
70
71 void BoxMoller (double d1, double d2, double *z1, double *z2)       //d1 y d2
      deben ser dos numeros con probabilidad plana; z1 y z2 son dos numeros
      devueltos con probabilidad gaussiana.
72 {
73     *z1 = sqrt(-2*log(d1)) * cos(2*pi*d2);
74     *z2 = sqrt(-2*log(d1)) * sin(2*pi*d2);
75 }
76
77 main()
78 {
79     double delta[tamanno_red], repressor[tamanno_red];
80     int i, j, k;
81     FILE *fout;
```

```
82
83      //Inicializar
84      ini_ran(123456789);
85      for(i=0;i<tamanno_red;i++)   //Bucle de inicializacion de cada celula
86      {
87          delta[i]=0.0;
88          repressor[i]=0.0;
89      }
90
91      construir_matriz_conectividad(M_conectividad);
92      epsilon = sqrt(t_step*2.0*Temperatura); //Se calcula una vez y vale para
            todo el programa. OJO si se modifica la T dinamicamente...
93
94      //El formato del fichero de salida es el siguiente:
95      //En la primera linea se dan los parametros utilizados.
96      //A continuacion se escribe en cada paso el estado global, que viene dado
            asi:
97      //En cada linea se dan los valores de cada celula: coordenadas i j en la red
            , valor de delta y de notch
98      fout=fopen("data.txt","wt");    //Abre el fichero e incluye los parametros
            en una primera linea
99      fprintf(fout, "#nu=%d\tbeta_d=%d\tbeta_r=%d\th=%d\tm=%d\tt_step=%lf\tN_pasos
            =%d\ttamanno_red=%d\tTemperatura=%lf\n#Nc=%d\tNf=%d\n", nu, beta_d,
            beta_r, h, m, t_step,N_pasos,tamanno_red, Temperatura, Nc, Nf);
100     for(i=0; i<Nf; i++)    //Escribe el estado inicial del sistema
101         for (j=0; j<Nc; j++)
102             fprintf(fout, "%d\t%d\t%lf\t%lf\n", j, i, delta[j+i*Nc], repressor[j
                    +i*Nc]);
103
104     for(k=0;k<N_pasos;k++)
105     {
106         pasoHeun(delta,repressor);
107         if(k%100==0)    //Guarda los datos en fichero cada 100 pasos
108             for(i=0; i<Nf; i++)
109                 for (j=0; j<Nc; j++)
110                     fprintf(fout, "%d\t%d\t%lf\t%lf\n", j, i, delta[j+i*Nc],
                            repressor[j+i*Nc]);
111     }
112
113     for(i=0; i<Nf; i++) //Guarda los datos del estado final
114         for (j=0; j<Nc; j++)
115             fprintf(fout, "%d\t%d\t%lf\t%lf\n", j, i, delta[j+i*Nc], repressor[j
                    +i*Nc]);
116 }
117
118 double frepressor(double repressor_i, double delta_j) //ecuacion diferencial
        para la actividad del repressor
119 {
120     return (beta_r * pow(delta_j,m)) / (1+pow(delta_j,m)) - repressor_i;
121 }
122
123 double fdelta(double repressor_i, double delta_i)//ecuacion diferencial para la
        actividad de delta
```

```
124 {
125     return nu * (beta_d/(1+pow(repressor_i,h)) - delta_i);
126 }
127
128 double grepressor(double repressor_i, double delta_j) //Parte que multiplica al
    ruido en la ecuacion del repressor
129 {
130     return sqrt(0.5*((beta_r * pow(delta_j,m)) / (1+pow(delta_j,m)) +
        valor_absoluto(repressor_i)));
131 }
132
133 double gdelta(double repressor_i, double delta_i)//Parte que multiplica al ruido
     en la ecuacion de delta
134 {
135     return sqrt(0.5*(nu * (beta_d/(1+pow(repressor_i,h)) + valor_absoluto(
        delta_i))));
136 }
137
138 void pasoHeun(double *delta, double *repressor) //Hace un paso de la simulacion
    con el algoritmo de Heun
139 {
140     double predictor_repressor[tamanno_red], predictor_delta[tamanno_red];
141     double z1_repressor[tamanno_red], z2_delta[tamanno_red];  //Dos numeros
        aleatorios que se obtienen por Box-Moller, para cada celula
142     double delta_vecinos, predicted_delta_vecinos; //Variable en la que se
        vuelca el valor promedio de delta de los vecinos de una celula
143     int i;
144
145     for (i=0; i<tamanno_red; i++)   //Generar los numeros aleatorios
146     {
147         BoxMoller(ParisiRapuano(), ParisiRapuano(), z1_repressor+i, z2_delta+i);
148     }
149
150     for (i=0; i<tamanno_red; i++)   //Calcular el primer paso (predictor)
151     {
152         delta_vecinos = promediar_delta_vecinos(delta, i);
153         predictor_repressor[i] = repressor[i] +
154                         frepressor(repressor[i], delta_vecinos) * t_step +
155                         grepressor(repressor[i], delta_vecinos) * epsilon *
                            z1_repressor[i];
156         predictor_delta[i] = delta[i] +
157                         fdelta(repressor[i], delta[i]) * t_step +
158                         gdelta(repressor[i], delta[i]) * epsilon * z2_delta[i];
159     }
160
161     for (i=0; i<tamanno_red; i++)   //Calcular el segundo paso
162     {
163         delta_vecinos = promediar_delta_vecinos(delta, i);
164         predicted_delta_vecinos = promediar_delta_vecinos(predictor_delta, i);
165
166         repressor[i] = repressor[i] +
167             0.5*(frepressor(repressor[i], delta_vecinos) +
```

```
168                 frepressor(predictor_repressor[i], predicted_delta_vecinos)) *
                        t_step
169             + 0.5*(grepressor(repressor[i], delta_vecinos) +
170                 grepressor(predictor_repressor[i], predicted_delta_vecinos)) *
                        epsilon * z1_repressor[i];
171         delta[i] = delta[i] + 0.5*(fdelta(repressor[i], delta[i]) +
172                             fdelta(predictor_repressor[i], predictor_delta[i])) *
                                t_step
173                                 + 0.5*(gdelta(repressor[i], delta[i]) +
174                             gdelta(predictor_repressor[i], predictor_delta[i])) *
                                epsilon * z2_delta[i];
175     }
176 }
177
178 double valor_absoluto (double x)
179 {   if(x<0.0) return -x;    else return x;  }
180
181 void construir_matriz_conectividad(bool M[tamanno_red][tamanno_red])
182 {
183     //En cada fila i, la matriz de conectividad dice que celulas se tocan con la
            celula i.
184     int i, j, avance;
185
186     //Primero se hacen cero todos los elementos.
187     for (i=0; i<tamanno_red; i++)
188         for (j=0; j<tamanno_red; j++)
189             M[i][j]=0;
190
191     for (i=0; i<tamanno_red; i++)   //Se van poniendo los 1s de la matriz,
            teniendo en cuenta los elementos de la frontera
192     {   //Se empieza con las esquinas
193         if (i==0)   //Esquina inferior izquierda
194         {
195             M[i][1]=1;
196             M[i][tamanno_red-1]=1;
197             M[i][Nc]=1;
198             M[i][tamanno_red-Nc]=1;
199             M[i][tamanno_red-Nc+1]=1;
200             M[i][Nc-1]=1;
201         }
202         else if (i==Nc-1)   //Esquina inferior derecha
203         {
204             M[i][0]=1;
205             M[i][Nc-1-1]=1;
206             M[i][Nc-1+Nc]=1;
207             M[i][Nc-1+1]=1;
208             M[i][tamanno_red-1]=1;
209             M[i][Nc-1+Nc-1]=1;
210         }
211         else if (i==tamanno_red-Nc) //Esquina superior izquierda
212         {
213             M[i][0]=1;
214             M[i][tamanno_red-Nc+1]=1;
```

```
215             M[i][tamanno_red-Nc-1]=1;
216             M[i][tamanno_red-Nc-Nc]=1;
217             M[i][tamanno_red-1]=1;
218             M[i][tamanno_red-Nc-Nc+1]=1;
219         }
220         else if (i==tamanno_red-1)  //Esquina superior derecha
221         {
222             M[i][0]=1;
223             M[i][tamanno_red-1-1]=1;
224             M[i][tamanno_red-1-Nc]=1;
225             M[i][Nc-1]=1;
226             M[i][Nc-1-1]=1;
227             M[i][tamanno_red-Nc]=1;
228         }
229         //Ahora se hacen los bordes
230         else if (i%Nc==0)  //Borde izquierdo
231         {
232             M[i][i+1]=1;
233             M[i][i-1]=1;
234             M[i][i+Nc]=1;
235             M[i][i-Nc]=1;
236             M[i][i-1+Nc]=1;
237             M[i][i-Nc+1]=1;
238         }
239         else if ((i+1)%Nc==0)  //Borde derecho
240         {
241             M[i][i+1]=1;
242             M[i][i-1]=1;
243             M[i][i+Nc]=1;
244             M[i][i-Nc]=1;
245             M[i][i+Nc-1]=1;
246             M[i][i-Nc+1]=1;
247         }
248         //El resto utiliza un algoritmo general
249         else
250         {
251             avance = i+1;
252             if(avance>=tamanno_red)
253                 avance -= tamanno_red;
254             M[i][avance] = 1;
255 
256             avance = i-1;
257             if(avance<0)
258                 avance += tamanno_red;
259             M[i][avance] = 1;
260 
261             //Vecinos de la fila superior e inferior
262             avance = i+Nc;
263             if(avance>=tamanno_red)
264                 avance -= tamanno_red;
265             M[i][avance] = 1;
266 
267             avance = i-Nc;
```

```
268             if(avance<0)
269                 avance += tamanno_red;
270             M[i][avance] = 1;
271 
272             //Ahora, dependiendo de si se trata de celula par o impar en la fila
                    , sus otros dos vecinos estaran en la fila superior
273             //o inferior.
274             if((i%Nc)%2 == 0)   //Al reducir a primera fila es par -> sus
                    vecinos estan en la fila de debajo suyo.
275             {
276                 if(i%Nc==0)
277                     avance = i+Nc-1;
278                 else
279                     avance = i-Nc+1;
280                 if(avance<0)
281                     avance += tamanno_red;
282                 M[i][avance] = 1;
283 
284                 avance = i-Nc-1;
285                 if(avance<0)
286                     avance += tamanno_red;
287                 M[i][avance] = 1;
288             }
289             else if((i%Nc)%2 == 1)   //Al reducir a primera fila es impar -> sus
                     vecinos estan en la fila de encima suyo.
290             {
291                 avance = i+Nc+1;
292                 if(avance>=tamanno_red)
293                     avance -= tamanno_red;
294                 M[i][avance] = 1;
295 
296                 avance = i+Nc-1;
297                 if(avance>=tamanno_red)
298                     avance -= tamanno_red;
299                 M[i][avance] = 1;
300             }
301             else
302                 printf("Ha ocurrido algo muy raro\n");
303         }
304     }
305 }
306 
307 double promediar_delta_vecinos (double *delta, int i_cell)
308 {
309     int i;
310     double delta_promedio;
311 
312     delta_promedio=0.0;
313     for (i=0; i<tamanno_red; i++)
314         if(M_conectividad[i][i_cell])
315             delta_promedio+=delta[i];
316 
317     return delta_promedio/(1.0*n_primeros_vecinos);
```

```
}
```

E.2. GLM_delta_notch

```
   ,
//LIBRERIAS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "cell_class.h"
#include "random_numbers.h"
#include "GLM_gif_fuerzas_hexatic.h"
#include "Algoritmos_para_minimizar.h"
#include "Modelo_Delta_Notch.h"

#define t_step_crecimiento 0.00001
#define t_step_movimiento 0.00001
#define factor_crecimiento 50.0
#define amplitud_ruido_crecimiento 50000000.0
#define amplitud_ruido_movimiento 70.0
#define cubic_root2 1.259921    //Raiz cubica de 2
#define max_cells 5000  //Limite maximo al numero de celulas que se puedan
    producir
#define R_umbral 1.0
#define N_pasos 100000
#define cte_pot_armonico 30000.0
#define R_equilibrio 1.95//2.5*R_umbral
#define R_cutoff 2.2//*R_umbral

//Parametros de la representacion
#define bordes 20.0
#define f_escala_fuerzas 0.00005    //Escala las fuerzas en la representacion
#define frecuencia_de_ploteo 10 //Cada cuantos pasos se hace el plot
#define plot_mode 2  //0 para ningun parametro representado, 1 para el parametro
     hexatico, 2 para delta y 3 para notch

#define frecuencia_de_guardado 10
//#define division_asimetrica //Segun este o no comentado, el modo es division
    simetrica (Inhibicion Lateral, IL) o asim. (DA)
#define N_celulas_finales 300   //Cuando llega a este numero de celulas, se para
#define N_pasos_adicionales 6000    //Una vez se ha parado, hace estos pasos
    para estabilizar delta y notch.

void crecer_celula (celula *cell);
void dividir_celula (celula *madre, celula *hija, int *lastID, double phi);
double potencia (double x, int power);
double Potencial_Armonico (double x1, double x2, double y1, double y2);
//devuelve el modulo de la fuerza entre dos fuerzas, a partir de sus coordenadas
void mover_celulas (celula *cell, int counter);
double potencial_hijas (celula *cells, int indice_madre, double theta);
void calculo_parametro_hexatico(celula cell[max_cells]);
```

```
44
45 main()
46 {
47     celula *cell;
48     int i, j, k, last_ID, indice_celula_hija, last_index, vecinos_division;
49     FILE * fout, *f_fuerzas, *f_hexatico, *f_deltanotch, *f_final_sim;
50     bool sennal_fin_division;   //Se hace 1 si se ha acabado de dividir, es
           decir, se ha llegado a N_celulas_finales
51
52     //Variables y objetos necesarios para la division
53     Golden golden;
54     double bracket_a, bracket_b, theta;   //angulo empleado para dividir las
           celulas y extremos del segmento de minimizacion
55
56     cell = new celula[max_cells];
57
58     ini_ran(clock());
59     fout = fopen("dataR.txt", "wt");
60     fprintf(fout, "#Max_cells = %d\tt_step_movimiento = %lf\tt_step_crecimiento
           = %lf\tN_pasos = %d\tN_pasos_adicionales = %d\tfrecuencia_de_guardado =
           %d\n", max_cells, t_step_movimiento, t_step_crecimiento, N_pasos,
           N_pasos_adicionales, frecuencia_de_guardado);
61     fprintf(fout, "#Factor_crecimiento = %lf\tamplitud_ruido_crecimiento = %lf\
           tamplitud_ruido_movimiento = %lf\tR_umbral = %lf\n",
62                     factor_crecimiento, amplitud_ruido_crecimiento,
                           amplitud_ruido_movimiento, R_umbral);
63     fprintf(fout, "#cte_pot_armonico = %lf\tR_equilibrio = %lf\n",
64                     cte_pot_armonico, R_equilibrio);
65
66     f_fuerzas = fopen("fuerzas.txt", "wt");
67     fclose(f_fuerzas);
68
69     f_hexatico = fopen("parametro_hexatico.txt", "wt");
70     f_deltanotch = fopen("delta_notch.txt", "wt");
71     f_final_sim = fopen("final_sim.txt", "wt"); //Un fichero que guarda los
           parametros y el tiempo en que acaba el crecimiento
72
73     fprintf(f_final_sim, "#Max_cells = %d\tt_step_movimiento = %lf\
           tt_step_crecimiento = %lf\tN_pasos = %d\tN_pasos_adicionales = %d\
           tfrecuencia_de_guardado = %d\n", max_cells, t_step_movimiento,
           t_step_crecimiento, N_pasos, N_pasos_adicionales, frecuencia_de_guardado
           );
74     fprintf(f_final_sim, "#Factor_crecimiento = %lf\tamplitud_ruido_crecimiento
           = %lf\tamplitud_ruido_movimiento = %lf\tR_umbral = %lf\n",
75                     factor_crecimiento, amplitud_ruido_crecimiento,
                           amplitud_ruido_movimiento, R_umbral);
76     fprintf(f_final_sim, "#cte_pot_armonico = %lf\tR_equilibrio = %lf\n",
77                     cte_pot_armonico, R_equilibrio);
78
79     //|||||||||||||||||||| CONDICIONES INICIALES ||||||||
80     for(i=0; i<max_cells; i++)
81     {
82         cell[i].inicializar_a_cero();
```

```
83          cell[i].R = 0.95;
84      }
85      cell[0].ID = 1; cell[1].ID = 2; //Dos celulas en contacto
86      cell[1].x = R_equilibrio;   last_ID = 2;
87      sennal_fin_division = 0;
88
89      //|||||||||||||||||||||| SIMULACION ||||||||||||
90      for(i=0; i<N_pasos; i++)
91      {
92          last_index = get_first_void_index(cell);
93          //CRECIMIENTO
94          for (j=0; j<last_index; j++)
95          {
96              if (cell[j].ID!=0 && cell[j].R <= R_umbral)
97              {
98                  crecer_celula(&cell[j]);
99              }
100         }
101         //MITOSIS
102         if(get_first_void_index(cell)<N_celulas_finales)     //la mitosis se
                detiene si hay N_celulas_finales celulas
103         for (j=0; j<last_index; j++)
104         {
105             if (cell[j].ID!=0 && cell[j].R>=R_umbral)
106             {
107                 indice_celula_hija = last_index;
108                 if (indice_celula_hija>max_cells)
109                     printf("\nNo hay memoria para dividir la celula. last_ID = %
                        d\n\n", last_ID);
110                 else
111                 {
112                     vecinos_division = 0;       //Se comprueba que no haya mas
                        de seis vecinos en el entorno de las celulas
113                     for(k=0; k<last_index; k++)
114                         if(distancia_entre_celulas(cell[j], cell[k])<=R_cutoff)
115                             vecinos_division++;
116                     if(vecinos_division<6)
117                     {
118                         //Se busca el minimo del potencial para decidir en que
                            direccion hacer la division
119                         Golden golden;
120                         golden.bracket(0.0, 3.14, potencial_hijas, cell, j);
121                         theta=golden.minimize(potencial_hijas, cell, j);
122                         dividir_celula(&cell[j], &cell[indice_celula_hija], &
                            last_ID, theta);
123                     }
124                 }
125             }
126         }
127         //MOVIMIENTO
128         for (j=0; j<t_step_crecimiento/t_step_movimiento;j++)   //El tiempo real
               se conserva, aunque G y M se integren con dif t_step
129             mover_celulas(cell, j);
```

```
130
131         //Calculo del parametro hexatico
132         calculo_parametro_hexatico(cell);
133
134         //Paso del modelo de Delta-Notch
135         pasoHeun(cell);
136
137         //Guardar los datos.
138         if(i%frecuencia_de_guardado==0){
139         for (j=0; j<last_index; j++)
140         {
141             if (cell[j].ID!=0)
142             {
143                 fprintf(fout, "%d\t%lf\t%lf\t%lf\t%lf\n", j, i*
                        t_step_crecimiento, cell[j].x, cell[j].y, cell[j].R);
144                 fprintf(f_hexatico, "%d\t%lf\n", j, sqrt(cell[j].
                        Re_parametro_hexatico*cell[j].Re_parametro_hexatico + cell[j
                        ].Im_parametro_hexatico*cell[j].Im_parametro_hexatico));
145                 fprintf(f_deltanotch, "%d\t%lf\t%lf\n", j, cell[j].delta, cell[j
                        ].repressor);
146             }
147         }
148         fprintf(fout, "%d\t%lf\t%lf\t%lf\t%lf\n", -1, i*t_step_crecimiento,
                -1.0, -1.0, -1.0);  //Linea de control: aqui acaba el repaso a las
                celulas en este instante de tiempo
149         }
150         if(i%100==0 && sennal_fin_division==0)printf("Completado %lf por ciento
                .\t%d pasos\r", (1.0*i)/(1.0*N_pasos)*100, i);
151         if(i%100==0 && sennal_fin_division==1)printf("Completado %lf por ciento
                .\t%d pasos\tPasos adicionales\r", (1.0*(i-N_pasos+
                N_pasos_adicionales))/(1.0*N_pasos_adicionales)*100, i);
152
153         if(sennal_fin_division==0 && get_first_void_index(cell)==
                N_celulas_finales-1)
154         {
155             fprintf(f_final_sim, "\nTiempo del final de crecimiento: %d\n", i);
156             sennal_fin_division = 1;printf("\n\nHa terminado de crecer, %d
                    celulas en %d pasos.\n\n", N_celulas_finales, i);
157             i = N_pasos - N_pasos_adicionales;
158         }
159     }
160     fclose(f_final_sim);
161     printf("\n\nNumero de celulas totales: %d\n\nNumero de celulas finales: %d\n
            \nComienza la representacion grafica.\n",
162             last_ID, get_first_void_index(cell));
163     dibujar_celulas_animacion_gif(bordes, f_escala_fuerzas, frecuencia_de_ploteo
            , plot_mode);
164     delete[] cell;
165 }
166
167 void crecer_celula (celula *cell)
168 {
169     double g1, g2;  double z;
```

```
170
171     BoxMoller(ParisiRapuano(), ParisiRapuano(), &z, &z);
172
173     g1 = factor_crecimiento * (cell->R + sqrt(amplitud_ruido_crecimiento *
            t_step_crecimiento) * z);
174     g2 = factor_crecimiento * (cell->R + t_step_crecimiento * g1);
175
176     cell->R += 0.5*t_step_crecimiento * (g1+g2);
177 }
178
179 void mover_celulas (celula *cell, int counter)
180 {
181     double *xvector, *yvector, *g1x, *g1y, *g2x, *g2y, *zx, *zy;
182     int i, j, last_cell;
183     double componente_de_la_fuerza, theta; //Esta variable permite hacer algunos
            calculos menos.
184     FILE *f_fuerzas;
185
186     xvector = new double[max_cells];    yvector = new double[max_cells];
187     g1x = new double[max_cells];        g2x = new double[max_cells];
188     g1y = new double[max_cells];        g2y = new double[max_cells];
189     zx = new double[max_cells];         zy = new double[max_cells];
190
191     f_fuerzas = fopen("fuerzas.txt", "at");
192     //Buscar la ultima celula existente
193     for (i=0; i<max_cells; i++)
194     {
195         if(cell[i].ID==0) //Solo hay celulas hasta este indice
196         {   last_cell=i-1;    i=max_cells+2;    }
197     }
198     //Generar todos los numeros aleatorios que voy a necesitar
199     for(i=0; i<=last_cell; i++)
200     {
201         BoxMoller(ParisiRapuano(), ParisiRapuano(), &zx[i], &zy[i]);
202     }
203     //Construir ahora los vectores x e y
204     for (i=0; i<=last_cell; i++)
205     {
206         xvector[i]=cell[i].x;
207         yvector[i]=cell[i].y;
208     }
209     //Ahora hay que calcular los g1
210     for (i=0; i<=last_cell; i++)
211     {
212         g1x[i]=g1y[i]=0.0;  //Primero inicializarlos a cero.
213         for (j=0; j<=last_cell; j++)
214         {
215             if (j!=i)
216             {
217                 componente_de_la_fuerza = Potencial_Armonico(
218     xvector[i] + sqrt(amplitud_ruido_movimiento * t_step_movimiento) * zx[i],
219     xvector[j] + sqrt(amplitud_ruido_movimiento * t_step_movimiento) * zx[j],
220     yvector[i] + sqrt(amplitud_ruido_movimiento * t_step_movimiento) * zy[i],
```

```
221     yvector[j] + sqrt(amplitud_ruido_movimiento * t_step_movimiento) * zy[j]);
222                 if (componente_de_la_fuerza!=0.0)
223                 {
224                     theta = atan2((yvector[j]-yvector[i]),(xvector[j]-xvector[i
                        ]));
225                     g1x[i]+=componente_de_la_fuerza*cos(theta);
226                     g1y[i]+=componente_de_la_fuerza*sin(theta);
227                 }
228             }
229         }
230         //if (counter%100==0)
231             fprintf(f_fuerzas, "%d\t%lf\t%lf\n", i, g1x[i], g1y[i]);
232     }//if (counter%100==0)
233         fprintf(f_fuerzas, "%d\t%lf\t%lf\n", -1, -1.0, -1.0);
234 
235     //A continuacion se calculan los g2
236     for (i=0; i<=last_cell; i++)
237     {
238         g2x[i]=g2y[i]=0.0;  //Primero inicializarlos a cero.
239         for (j=0; j<=last_cell; j++)
240         {
241             if (j!=i)
242             {
243                 componente_de_la_fuerza = Potencial_Armonico(
244                 xvector[i] + t_step_movimiento * g1x[i],
245                 xvector[j] + t_step_movimiento * g1x[j],
246                 yvector[i] + t_step_movimiento * g1y[i],
247                 yvector[j] + t_step_movimiento * g1y[j]);
248                 if (componente_de_la_fuerza!=0.0)
249                 {
250                  theta=atan2((yvector[j]-yvector[i]),(xvector[j]-xvector[i]));
251                  g2x[i]+=componente_de_la_fuerza*cos(theta);
252                  g2y[i]+=componente_de_la_fuerza*sin(theta);
253                 }
254             }
255         }
256     }
257     //Finalmente, se suman los valores finales
258     for (i=0; i<=last_cell; i++)
259     {
260         cell[i].x += 0.5*t_step_movimiento * (g1x[i] + g2x[i]) + sqrt(
                amplitud_ruido_movimiento * t_step_movimiento) * zx[i];
261         cell[i].y += 0.5*t_step_movimiento * (g1y[i] + g2y[i]) + sqrt(
                amplitud_ruido_movimiento * t_step_movimiento) * zy[i];
262     }
263     fclose(f_fuerzas);
264     delete[] xvector;  delete[] yvector; delete[] g1x;   delete[] g1y;
265     delete[] g2x;  delete[] g2y;  delete[] zx;   delete[] zy;
266 }
267 
268 double potencia (double x, int power)
269 {//Implementacion eficiente de una potencia de exponente entero
270     int i;
```

```
271     double resultado;
272     resultado=1.0;
273     for (i=0; i<power; i++) resultado*=x;
274     return resultado;
275 }
276
277 double Potencial_Armonico (double x1, double x2, double y1, double y2)   //
        devuelve el modulo de la fuerza entre dos celulas, a partir de sus
        coordenadas
278 {
279     double distancia;
280     distancia = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
281     if(distancia>2.4*R_umbral)  return 0.0;
282     else    return cte_pot_armonico*(distancia-R_equilibrio);
283 }
284
285 void dividir_celula (celula *madre, celula *hija, int *lastID, double phi)
286 {
287     int i;
288
289     for (i=0; i<max_generaciones; i++)
290     {
291         if (madre->arbol_genealogico[i]==0)
292         {
293             madre->arbol_genealogico[i]=madre->ID;
294             i=max_generaciones+2;
295         }
296     }
297
298     if(i<max_generaciones+1)
299     {
300         printf("\nSe han agotado las generaciones disponibles. \nID mas grande:
                %d\n", *lastID);
301     }
302
303     madre->R/=cubic_root2;   //Se parte el volumen
304     hija->copiar_celula(*madre);  //Se copia la celula para modificar las
            propiedades a partir de las suyas
305
306     madre->x+=madre->R*cos(phi);
307     madre->y+=madre->R*sin(phi);
308     hija->x-=hija->R*cos(phi);
309     hija->y-=hija->R*sin(phi);
310
311     madre->ID = *lastID + 1;
312     hija->ID = *lastID + 2;
313
314     *lastID += 2;
315
316     #ifdef division_asimetrica
317     //Division asimetrica de Delta-Notch:
318     madre->delta=0.0; //madre->repressor*=2.0;
319     hija->repressor=0.0; //hija->delta*=2.0;
```

```
320     #endif // division_asimetrica
321 }
322
323 //La siguiente funcion calcula el potencial al que estaran sometidas
324 //las dos celulas hijas si una determinada celula se divide,
325 //de forma que las hijas se situen con un angulo theta
326 double potencial_hijas (celula *cells, int indice_madre, double theta)
327 {
328     int i, last_index;
329     double proyeccionx, proyecciony; //son las distancias que se separan las
        celulas hija respecto de la madre.
330     double *distancias_1, *distancias_2;
331     double potencial;
332
333     distancias_1 = new double[max_cells];   distancias_2 = new double[max_cells
        ];
334
335     last_index = get_first_void_index(cells);
336
337     //Calculo de las proyecciones
338     proyeccionx = cells[indice_madre].R * cos(theta);
339     proyecciony = cells[indice_madre].R * sin(theta);
340
341     for(i=0; i<last_index; i++)
342     {
343         if (i==indice_madre)    i++;    //Se salta este caso porque la celula
            madre no existiria tras la division y por tanto no hay que contarla
344         if(cells[i].ID != 0)
345         {
346     distancias_1[i] = sqrt( (cells[indice_madre].x + proyeccionx - cells[i].x) *
347                             (cells[indice_madre].x + proyeccionx - cells[i].x)
348                            +(cells[indice_madre].y + proyecciony - cells[i].y) *
349                             (cells[indice_madre].y + proyecciony - cells[i].y));
350     distancias_2[i] = sqrt( (cells[indice_madre].x - proyeccionx - cells[i].x) *
351                             (cells[indice_madre].x - proyeccionx - cells[i].x)
352                            +(cells[indice_madre].y - proyecciony - cells[i].y) *
353                             (cells[indice_madre].y - proyecciony - cells[i].y));
354             if (distancias_1[i] > R_cutoff) distancias_1[i] = 0.0;
355             if (distancias_2[i] > R_cutoff) distancias_2[i] = 0.0;
356         }
357         else
358         { distancias_1[i] = 0.0;  distancias_2[i] = 0.0; }
359     }
360     //Se hacen cero las distancias a la madre, para anular la contribucion.
361     distancias_1[indice_madre] = 0.0;   distancias_2[indice_madre] = 0.0;
362     //A continuacion se procede a hacer la suma de todas las contribuciones.
363     potencial = 0.0;
364     for(i=0; i<last_index; i++)
365     {
366         potencial += 0.5 * cte_pot_armonico * (distancias_1[i]*distancias_1[i] +
367         distancias_2[i]*distancias_2[i]) - cte_pot_armonico * R_equilibrio *
368                                           (distancias_1[i] + distancias_2[i]);
369     }
```

```
370     delete[] distancias_1; delete[] distancias_2;
371     return potencial;
372 }
373
374 void calculo_parametro_hexatico(celula *cell)
375 {
376     double *delta_X, *delta_Y, *distancia;
377     int i, j, last_index, numero_vecinos;
378     double seno, coseno;
379
380     delta_X = new double[max_cells*max_cells];
381     delta_Y = new double[max_cells*max_cells];
382     distancia = new double[max_cells*max_cells];
383
384     //Se busca primero el ultimo indice empleado en cell
385     for (i=0; i<max_cells; i++)
386     {
387         if(cell[i].ID==0)
388         {
389             last_index=i;
390             i=max_cells+2;
391         }
392     }
393
394     //Si solo hay una celula, el parametro no tiene sentido y se devuelven solo
        ceros.
395     if(last_index==1)
396     {
397         cell[0].Re_parametro_hexatico = 0.0;
398         cell[0].Im_parametro_hexatico = 0.0;
399         delete[] delta_X;  delete[] delta_Y;  delete[] distancia;
400         return;
401     }
402
403     //Se calculan las distancias
404     for(i=0; i<last_index; i++) //La matriz es antisimetrica
405     {
406         for(j=0; j<last_index; j++)
407         {
408             *(delta_X + j*(last_index-1) + i) = cell[i].x - cell[j].x;
409             *(delta_Y + j*(last_index-1) + i) = cell[i].y - cell[j].y;
410             *(distancia + j*(last_index-1) + i) =
411             sqrt(*(delta_X + j*(last_index-1) + i)*
412                  *(delta_X + j*(last_index-1) + i) +
413                  *(delta_Y + j*(last_index-1) + i) *
414                  *(delta_Y + j*(last_index-1) + i));
415         }
416     }
417     //Se inicializan a cero los parametros
418     for(i=0; i<last_index; i++)
419     {
420         cell[i].Re_parametro_hexatico = 0.0;
421         cell[i].Im_parametro_hexatico = 0.0;
```

```
422     }
423     //Se va calculando el parametro para cada celula
424     for(i=0; i<last_index; i++)
425     {
426         numero_vecinos = 0;
427         for(j=0; j<last_index; j++)
428         {
429             if(*(distancia + j*(last_index-1) + i)<=R_cutoff && i!=j)
430             {
431                 coseno = *(delta_X + j*(last_index-1) + i)
432                 / *(distancia + j*(last_index-1) + i);
433                 seno = *(delta_Y + j*(last_index-1) + i)
434                 / *(distancia + j*(last_index-1) + i);
435                 cell[i].Re_parametro_hexatico +=
436                 potencia(coseno,6) -
437                 15.0 * potencia(coseno, 4) * potencia(seno, 2)
438                 + 15.0 * potencia(seno, 4) * potencia(coseno, 2) -
439                 potencia(seno,6);
440                 cell[i].Im_parametro_hexatico +=
441                 6.0 * potencia(coseno, 5) * seno -
442                 20.0 * potencia(coseno, 3) * potencia(seno, 3) +
443                 6.0 * potencia(seno, 5) * coseno;
444                 numero_vecinos++;
445             }
446         }
447         if(numero_vecinos==0)
448         {
449             cell[i].Re_parametro_hexatico = 0.0;
450             cell[i].Im_parametro_hexatico = 0.0;
451         }
452         else
453         {
454             cell[i].Re_parametro_hexatico /= (1.0*numero_vecinos);
455             cell[i].Im_parametro_hexatico /= (1.0*numero_vecinos);
456         }
457     }
458     delete[] delta_X;  delete[] delta_Y;  delete[] distancia;
459 }
```

E.3. cell_class

```
   ,
1  #define max_generaciones 200    //numero maximo de generaciones que pueda haber.
2
3  class celula
4  {
5  public:
6      double x;   //coordenada x
7      double y;   //coordenada y
8      double R;   //Radio, que da el volumen
9      double Re_parametro_hexatico;   //Parte real e imaginaria
10     double Im_parametro_hexatico;   //del parametro hexatico
```

```
11      double delta;   //concentraciones de Delta
12      double repressor;   //y repressor
13      unsigned int generacion; //generacion a la que pertenece la celula
14      unsigned int ID; //numero que identifica a cada celula individual
15      int arbol_genealogico[max_generaciones];    //Almacena el arbol filogenetico
            de la celula
16
17      void inicializar_a_cero(void)
18      {
19          int i;
20          x=y=R=Re_parametro_hexatico=Im_parametro_hexatico=delta=repressor=0.0;
21          generacion=0;
22          ID=0;
23          for(i=0; i<max_generaciones; i++)
24              arbol_genealogico[i]=0;
25      }
26
27      void copiar_celula(celula madre)
28      {
29          int i;
30          x=madre.x;  y=madre.y;  R=madre.R;
31          Re_parametro_hexatico=madre.Re_parametro_hexatico;
32          Im_parametro_hexatico=madre.Im_parametro_hexatico;
33          delta=madre.delta;  repressor=madre.repressor;
34          generacion=madre.generacion;
35          ID=madre.ID;
36          for(i=0; i<max_generaciones; i++)
37              arbol_genealogico[i]=madre.arbol_genealogico[i];
38      }
39  };
```

E.4. random_numbers

```
    ,
1   #define NormRANu (2.3283063671E-10F)  //Parisi-Rapuano
2   #define pi 3.141592     //La usa Box Moller
3   extern float ParisiRapuano(void);
4   extern void ini_ran(int seed);
5   void BoxMoller (double d1, double d2, double *z1, double *z2);
6
7   unsigned int irr[256];
8   unsigned int ir1;
9   unsigned char ind_ran, ig1, ig2, ig3;
10
11  extern void ini_ran(int seed)
12  {
13      int ini, factor, sum, i;
14      srand(seed);
15      ini = seed;
16      factor = 67397; sum = 7364893;
17      for(i=0; i<256; i++)
18      {
```

```
        ini = (ini*factor) + sum;
        irr[i] = ini;
    }
    ind_ran = ig1 = ig2 = ig3 = 0;
}

float ParisiRapuano(void)
{
    float r;
    ig1 = ind_ran-24;   ig2 = ind_ran-55;   ig3 = ind_ran-61;
    irr[ind_ran] = irr[ig1]+irr[ig2];
    ir1 = (irr[ind_ran]^irr[ig3]);
    ind_ran++;
    r = ir1*NormRANu;
    return r;
}

void BoxMoller (double d1, double d2, double *z1, double *z2)       //d1 y d2
    deben ser dos numeros con probabilidad plana; z1 y z2 son dos numeros
    devueltos con probabilidad gaussiana.
{
        *z1 = sqrt(-2*log(d1)) * cos(2*pi*d2);
        *z2 = sqrt(-2*log(d1)) * sin(2*pi*d2);
}
//Siempre se debe inicializar con el comando siguiente:
//      ini_ran(####);
```

E.5. Algoritmos_para_minimizar

```
   ,
#include <algorithm>
#include <cmath>
#define max_cells
/*Los algoritmos aqui presentados proceden del libro
Numerical Recipes (2007), concretamente del capitulo
10, secciones 10.1 y 10.2*/

class celula;
template <class T> void intercambio ( T& a, T& b );
template <class T> const T& maximo (const T& a, const T& b);
double double_abs (double x);

struct Bracketmethod { //mins.h
    //Base class for one-dimensional minimization routines. Provides a routine
        to bracket a minimum
    //and several utility functions.
    double ax,bx,cx,fa,fb,fc;
    template <class T>
    void bracket(const double a, const double b, T &func, celula *cell, int
        indice_madre)
    //Given a function or functor func, and given distinct initial points ax and
         bx, this routine
```

```
20      //searches in the downhill direction (defined by the function as evaluated
            at the initial points)
21      //and returns new points ax, bx, cx that bracket a minimum of the function.
            Also returned
22      //are the function values at the three points, fa, fb, and fc.
23      {
24          const double GOLD = 1.618034,GLIMIT = 100.0,TINY = 1.0e-20;
25          //Here GOLD is the default ratio by which successive intervals are
                magnified and GLIMIT
26          //is the maximum magnification allowed for a parabolic-fit step.
27          ax = a; bx = b;
28          double fu;
29          fa = func(cell, indice_madre, ax);
30          fb = func(cell, indice_madre, bx);
31          if (fb > fa) {    //Switch roles of a and b so that we can go
32              intercambio(ax,bx); //downhill in the direction from a to b.
33              intercambio(fb,fa);
34          }
35          cx = bx+GOLD*(bx-ax); //First guess for c.
36          fc = func(cell, indice_madre, cx);
37          while (fb > fc) //Keep returning here until we bracket.
38          {
39              double r = (bx-ax)*(fb-fc); //Compute u by parabolic extrapolation
                    from
40                                          //a; b; c. TINY is used to prevent any
                                                possible
41                                          //division by zero.
42              double q = (bx-cx)*(fb-fa);
43              double u = bx-((bx-cx)*q-(bx-ax)*r) / (2.0*copysign(maximo(
                    double_abs(q-r),TINY),q-r));
44              double ulim = bx+GLIMIT*(cx-bx);
45              //We wont go farther than this. Test various possibilities:
46              if ((bx-u)*(u-cx) > 0.0) { //Parabolic u is between b and c: try it.
47                  fu = func(cell, indice_madre, u);
48                  if (fu < fc) { //Got a minimum between b and c.
49                      ax = bx; bx = u; fa = fb; fb = fu;
50                      return;
51                  } else if (fu > fb) { //Got a minimum between between a and u.
52                      cx = u;
53                      fc = fu;
54                      return;
55                  }
56                  u = cx+GOLD*(cx-bx); //Parabolic fit was no use. Use default
                        magnification.
57                  fu = func(cell, indice_madre, u);
58              } else if ((cx-u)*(u-ulim) > 0.0) { //Parabolic fit is between c and
                     its allowed limit.
59                  fu = func(cell, indice_madre, u);
60                  if (fu < fc) {
61                      shft3(bx,cx,u,u+GOLD*(u-cx));
62                      shft3(fb,fc,fu,func(cell, indice_madre, u));
63                  }
```

```
64              } else if ((u-ulim)*(ulim-cx) >= 0.0) { //Limit parabolic u to
                    maximum allowed value.
65                  u = ulim;
66                  fu = func(cell, indice_madre, u);
67              } else { //Reject parabolic u, use default magnification.
68                  u = cx+GOLD*(cx-bx);
69                  fu = func(cell, indice_madre, u);
70              }
71              shft3(ax,bx,cx,u); //Eliminate oldest point and continue.
72              shft3(fa,fb,fc,fu);
73          }
74      }
75      inline void shft2(double &a, double &b, const double c)
76      //Utility function used in this structure or others derived from it.
77      { a = b;  b = c; }
78      inline void shft3(double &a, double &b, double &c, const double d)
79      { a = b;  b = c;  c = d; }
80      inline void mov3(double &a, double &b, double &c, const double d, const
            double e, const double f)
81      { a = d; b = e; c = f; }
82  };
83
84
85  struct Golden : Bracketmethod { //mins.h
86      //Golden section search for minimum.
87      double xmin,fmin;
88      const double tol;
89      Golden(const double toll = 3.0e-8) : tol(toll) {}
90      template <class T>
91
92      double minimize(T &func, celula *cell, int indice_madre)
93      //Given a function or functor f, and given a bracketing triplet of abscissas
            ax, bx, cx (such
94      //that bx is between ax and cx, and f(bx) is less than both f(ax) and f(cx))
            , this routine
95      //performs a golden section search for the minimum, isolating it to a
            fractional precision of
96      //about tol. The abscissa of the minimum is returned as xmin, and the
            function value at
97      //the minimum is returned as min, the returned function value.
98      {
99          const double R = 0.61803399, C = 1.0-R; //The golden ratios.
100         double x1,x2;
101         double x0 = ax; //At any given time we will keep track of four
102         double x3 = cx; //points, x0,x1,x2,x3.
103         if (abs(cx-bx) > abs(bx-ax)) { //Make x0 to x1 the smaller segment,
104             x1 = bx;
105             x2 = bx+C*(cx-bx); //and fill in the new point to be tried.
106         } else {
107             x2 = bx;
108             x1 = bx-C*(bx-ax);
109         }
110
```

```
111         double f1 = func(cell, indice_madre, x1); //The initial function
                evaluations. Note that
112                 //we never need to evaluate the function
113                 //at the original endpoints.
114         double f2 = func(cell, indice_madre, x2);
115 
116         while (abs(x3-x0) > tol*(abs(x1)+abs(x2))) {
117             if (f2 < f1) { //One possible outcome,
118                 shft3(x0,x1,x2,R*x2+C*x3); //its housekeeping,
119                 shft2(f1,f2,func(cell, indice_madre, x2)); //and a new function
                        evaluation.
120             } else {    //The other outcome,
121                 shft3(x3,x2,x1,R*x1+C*x0);
122                 shft2(f2,f1,func(cell, indice_madre, x1)); //and its new
                        function evaluation.
123             }
124         } //Back to see if we are done.
125         if (f1 < f2) { //We are done. Output the best of the two
126             xmin = x1; //current values.
127             fmin = f1;
128         } else {
129             xmin = x2;
130             fmin = f2;
131         }
132         return xmin;
133     }
134 };
135 
136 template <class T> void intercambio ( T& a, T& b )
137 {
138   T c(a); a=b; b=c;
139 }
140 
141 template <class T> const T& maximo (const T& a, const T& b) {
142   return (a<b)?b:a;     // or: return comp(a,b)?b:a; for version (2)
143 }
144 
145 double double_abs (double x)
146 {
147     if(x>=0)    return x;
148     else    return -x;
149 }
```

E.6. GLM_graficos

```
    ,
1 #include <stdio.h>
2 
3 #define GNUPLOT "D:\\Program\ Files\\gnuplot\\bin\\gnuplot.exe" //path de
      gnuplot
4 //#define representar_fuerzas
5 
```

```
6  int dibujar_celulas_animacion_gif(double bordes, double f_escala_fuerzas, int
       frecuencia_de_ploteo, int mode)
7  /*Modos:
8      0 - No pintar las celulas de modo alguno
9      1 - Pintar el parametro hexatico
10     2 - Pintar Delta
11     3 - Pintar Notch
12     */
13 {
14     int i, j, i_celula, max_cells, N_pasos, N_pasos_adicionales,
           frecuencia_de_guardado;
15     double x, y, R, time, t_step, dummy, fuerzax, fuerzay;
16     FILE *gp, *fentrada, *f_fuerzas, *f_hexatico, *f_deltanotch, *f_final_sim;
17     double hexatic, max_hexatic, min_hexatic;
18     double delta, max_delta, min_delta;
19     double notch, max_notch, min_notch;
20
21     //Se preparan los parametros para el hexatic parameter
22     if(mode==1)
23     {
24         f_hexatico = fopen("parametro_hexatico.txt", "rt");
25         max_hexatic = -9999;
26         min_hexatic = 9999;
27         while(!feof(f_hexatico))
28         {
29             fscanf(f_hexatico, "%d\t%lf\n", &i, &hexatic);
30             if(hexatic>max_hexatic)   max_hexatic = hexatic;
31             if(hexatic<min_hexatic)   min_hexatic = hexatic;
32         }
33         printf("\n\nMAX_hexatico = %lf\tMIN_hexatico = %lf\n\n", max_hexatic,
               min_hexatic);
34         fclose(f_hexatico);
35     }
36
37     if(mode==2)
38     {
39         f_deltanotch = fopen("delta_notch.txt", "rt");
40         max_delta = -9999;
41         min_delta = 9999;
42         while(!feof(f_deltanotch))
43         {
44             fscanf(f_deltanotch, "%d\t%lf\t%lf\n", &i, &delta, &notch);
45             if(delta>max_delta)   max_delta = delta;
46             if(delta<min_delta)   min_delta = delta;
47         }
48         printf("\n\nMAX_delta = %lf\tMIN_delta = %lf\n\n", max_delta, min_delta)
               ;
49         fclose(f_deltanotch);
50     }
51
52     if(mode==3)
53     {
54         f_deltanotch = fopen("delta_notch.txt", "rt");
```

```
55          max_notch = -9999;
56          min_notch = 9999;
57          while(!feof(f_deltanotch))
58          {
59              fscanf(f_deltanotch, "%d\t%lf\t%lf\n", &i, &delta, &notch);
60              if(notch>max_notch)   max_notch = notch;
61              if(notch<min_notch)   min_notch = notch;
62          }
63          printf("\n\nMAX_notch = %lf\tMIN_notch = %lf\n\n", max_notch, min_notch)
                ;
64          fclose(f_deltanotch);
65      }
66
67      //Ahora se abre la pipe a gnuplot
68      gp = popen("D:\\gnuplot\\bin\\gnuplot.exe","w");
69      if (gp==NULL)
70      {printf("Error al abrir la tuberia a gnuplot. \n");
71      return 0;}
72
73      fentrada=fopen("dataR.txt", "rt");   //Se reabre el archivo de entrada
74      #ifdef representar_fuerzas
75      f_fuerzas = fopen("fuerzas.txt", "rt");
76      #endif // representar_fuerzas
77      if(mode==1) f_hexatico = fopen("parametro_hexatico.txt", "rt");
78      if(mode==2 || mode==3)  f_deltanotch = fopen("delta_notch.txt", "rt");
79
80      //Se escanean los parametros que se han incluido en las primeras lineas del
            archivo de datos principal
81      fscanf(fentrada, "#Max_cells = %d\tt_step_movimiento = %lf\
            tt_step_crecimiento = %lf\tN_pasos = %d\tN_pasos_adicionales = %d\
            tfrecuencia_de_guardado = %d\n", &max_cells, &dummy, &t_step, &dummy, &
            N_pasos_adicionales, &frecuencia_de_guardado);
82      fscanf(fentrada, "#Factor_crecimiento = %lf\tamplitud_ruido_crecimiento = %
            lf\tamplitud_ruido_movimiento = %lf\tR_umbral = %lf\n",
83                      &dummy, &dummy, &dummy, &dummy);
84      fscanf(fentrada, "#cte_pot_armonico = %lf\tR_equilibrio = %lf\n",
85                      &dummy, &dummy);
86
87      //Se vuelve a tomar los parametros pero del archivo que incluye el final de
            las simulaciones.
88      f_final_sim = fopen("final_sim.txt", "rt");
89      fscanf(f_final_sim, "#Max_cells = %d\tt_step_movimiento = %lf\
            tt_step_crecimiento = %lf\tN_pasos = %d\tN_pasos_adicionales = %d\
            tfrecuencia_de_guardado = %d\n", &max_cells, &dummy, &t_step, &N_pasos,
            &N_pasos_adicionales, &frecuencia_de_guardado);
90      fscanf(f_final_sim, "#Factor_crecimiento = %lf\tamplitud_ruido_crecimiento =
             %lf\tamplitud_ruido_movimiento = %lf\tR_umbral = %lf\n",
91                      &dummy, &dummy, &dummy, &dummy);
92      fscanf(f_final_sim, "#cte_pot_armonico = %lf\tR_equilibrio = %lf\n",
93                      &dummy, &dummy);
94      fscanf(f_final_sim, "\nTiempo del final de crecimiento: %d\n", &N_pasos);
95      fclose(f_final_sim);
96
```

```
97      //Comandos de cabecera para el plot
98      fprintf(gp, "set terminal gif animate transparent enhanced font \"Verdana,
            22\" size 1500,1500\n");
99      fprintf(gp, "set output \"plot.gif\"\n");
100
101     fprintf(gp, "set xrange [%lf:%lf]\n", -bordes, bordes);
102     fprintf(gp, "set yrange [%lf:%lf]\n", -bordes, bordes);
103
104     //Escribe el modo en el grafico
105     if(mode==1) fprintf(gp, "set label 1 \"Parametro hexatico\" at graph 0,1\n")
            ;
106     if(mode==2) fprintf(gp, "set label 1 \"Delta\" at graph 0,1\n");
107     if(mode==3) fprintf(gp, "set label 1 \"Notch\" at graph 0,1\n");
108
109     //Un bucle que va haciendo cada instante del gif
110     for(j=0; j<(N_pasos/frecuencia_de_guardado + N_pasos_adicionales/
            frecuencia_de_guardado); j++)
111     {
112         for(i=0; i<max_cells; i++)
113         {
114             fscanf(fentrada, "%d\t%lf\t%lf\t%lf\t%lf\n", &i_celula, &time, &x, &
                    y, &R);
115             #ifdef representar_fuerzas
116             fscanf(f_fuerzas, "%d\t%lf\t%lf\n", &dummy, &fuerzax, &fuerzay);
117             #endif // representar_fuerzas
118             if(i_celula!=-1)
119             {   // fs solid
120                 if(mode==1)     fscanf(f_hexatico, "%d\t%lf\n", &dummy, &hexatic
                        );
121                 if(mode==1)     fprintf(gp, "set object %d circle at %lf, %lf
                        size %lf fc rgb \"#%X\" fs solid 0.5 linewidth 2.0\n",
                        i_celula+1, x, y, R, 11075584 + 256 * ((int) (255 * (1 - (
                        hexatic - min_hexatic)/max_hexatic))));
122                 if(mode==2 || mode==3)    fscanf(f_deltanotch, "%d\t%lf\t%lf\n"
                        , &dummy, &delta, &notch);
123                 if(mode==2)     fprintf(gp, "set object %d circle at %lf, %lf
                        size %lf fc rgb \"#%X\" fs solid 0.5 linewidth 2.0\n",
                        i_celula+1, x, y, R, 11075584 + 256 * ((int) (255 * (1 - (
                        delta - min_delta)/max_delta))));
124                 if(mode==3)     fprintf(gp, "set object %d circle at %lf, %lf
                        size %lf fc rgb \"#%X\" fs solid 0.5 linewidth 2.0\n",
                        i_celula+1, x, y, R, 11075584 + 256 * ((int) (255 * (1 - (
                        notch - min_notch)/max_notch))));
125                 fprintf(gp, "set arrow %d from %lf, %lf to %lf, %lf\n",
                        i_celula+1, x, y, x+fuerzax*f_escala_fuerzas, y+fuerzay*
                        f_escala_fuerzas);
126             }
127             else
128                 i=max_cells+2;
129         }
130         if(j%frecuencia_de_ploteo==0)
131             fprintf(gp, "plot 1/0 title''\n");  //Poner algo que al plotearlo no
                    se vea en el grafico final
```

```
132         printf("Graficado el %lf por ciento\r", 100*(1.0*j)/(1.0*(N_pasos+
                N_pasos_adicionales)/frecuencia_de_guardado));
133     }
134     fflush(gp);
135 
136     fclose(gp); fclose(fentrada); return 0;
137 }
```

E.7. modelo_delta_notch

```
   ,
1  //LIBRERIAS
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include <time.h>
6  
7  //CONSTANTES
8  #define max_cells 100000
9  #define R_cutoff 2.2
10 #define nu 1
11 #define beta_d 50
12 #define beta_r 50
13 #define h 4 //h y m son los coeficientes (exponentes) de Hill. Son exponentes
       que aparecen en la funcion de Hill.
14 #define m 4
15 #define t_step 0.001    //Paso temporal
16 //Constantes estocasticas
17 #define Temperatura 0.01
18 
19 class celula;
20 void pasoHeun(celula *cell); //Hace un paso de Runge Kutta
21 double frepressor(double repressor_i, double delta_j); //ecuacion diferencial
       para la actividad del repressor
22 double fdelta(double repressor_i, double delta_i);//ecuacion diferencial para la
        actividad de delta
23 double grepressor(double repressor_i, double delta_j); //Parte que multiplica al
        ruido en la ecuacion del repressor
24 double gdelta(double repressor_i, double delta_i); //Parte que multiplica al
       ruido en la ecuacion de delta
25 double valor_absoluto (double x);   //Implementacion de valor absoluto para
       doubles
26 void construir_matriz_conectividad(bool *M, celula *cell);
27 double promediar_delta_vecinos (celula *cell, int i_cell, bool *M);
28 double distancia_entre_celulas(celula cell1, celula cell2);
29 int get_first_void_index(celula *cell);
30 double promediar_delta_vecinos_vpredictor (celula *cell, double *delta_predictor
       , int i_cell, bool *M);
31 
32 double frepressor(double repressor_i, double delta_j) //ecuacion diferencial
       para la actividad del repressor
33 {
```

```
34      return (beta_r * pow(delta_j,m)) / (1+pow(delta_j,m)) - repressor_i;
35  }
36
37  double fdelta(double repressor_i, double delta_i)//ecuacion diferencial para la
        actividad de delta
38  {
39      return nu * (beta_d/(1+pow(repressor_i,h)) - delta_i);
40  }
41
42  double grepressor(double repressor_i, double delta_j) //Parte que multiplica al
        ruido en la ecuacion del repressor
43  {
44      return sqrt(0.5*((beta_r * pow(delta_j,m)) / (1+pow(delta_j,m)) +
            valor_absoluto(repressor_i)));
45  }
46
47  double gdelta(double repressor_i, double delta_i)//Parte que multiplica al ruido
        en la ecuacion de delta
48  {
49      return sqrt(0.5*(nu * (beta_d/(1+pow(repressor_i,h)) + valor_absoluto(
            delta_i))));
50  }
51
52  void pasoHeun(celula *cell) //Hace un paso de Heun
53  {
54      double *predictor_repressor, *predictor_delta;
55      double *z1_repressor, *z2_delta;  //Dos numeros aleatorios por Box-Moller
56      double delta_vecinos, predicted_delta_vecinos; //Variable en la que se
            vuelca el valor promedio de delta de los vecinos de una celula
57      double epsilon;
58      bool *M;
59      int i, last_index;
60
61      predictor_repressor = new double[max_cells]; predictor_delta = new double[
            max_cells];
62      z1_repressor = new double[max_cells]; z2_delta = new double[max_cells];
63      M = new bool[max_cells*max_cells];
64
65      //Se preparan los parametros: cual es el ultimo indice y crear la matriz de
            conectividad.
66      last_index = get_first_void_index(cell);
67      construir_matriz_conectividad(M,cell);
68      epsilon = sqrt(t_step*2.0*Temperatura);
69
70      for (i=0; i<last_index; i++)
71      {
72          BoxMoller(ParisiRapuano(), ParisiRapuano(), z1_repressor+i, z2_delta+i);
73      }
74
75      for (i=0; i<last_index; i++)
76      {
77          delta_vecinos = promediar_delta_vecinos(cell, i, M);
```

```
78          predictor_repressor[i] = cell[i].repressor + frepressor(cell[i].
                repressor, delta_vecinos) * t_step + grepressor(cell[i].repressor,
                delta_vecinos) * epsilon * z1_repressor[i];
79          predictor_delta[i] = cell[i].delta + fdelta(cell[i].repressor, cell[i].
                delta) * t_step + gdelta(cell[i].repressor, cell[i].delta) * epsilon
                 * z2_delta[i];
80      }
81
82      for (i=0; i<last_index; i++)
83      {
84          delta_vecinos = promediar_delta_vecinos(cell, i, M);
85          predicted_delta_vecinos = promediar_delta_vecinos_vpredictor(cell,
                predictor_delta, i, M);
86
87          cell[i].repressor = cell[i].repressor + 0.5*(frepressor(cell[i].
                repressor, delta_vecinos) + frepressor(predictor_repressor[i],
                predicted_delta_vecinos)) * t_step
88                                      + 0.5*(grepressor(cell[i].repressor,
                                            delta_vecinos) + grepressor(
                                            predictor_repressor[i],
                                            predicted_delta_vecinos)) * epsilon *
                                            z1_repressor[i];
89          cell[i].delta = cell[i].delta + 0.5*(fdelta(cell[i].repressor, cell[i].
                delta) + fdelta(predictor_repressor[i], predictor_delta[i])) *
                t_step
90                                  + 0.5*(gdelta(cell[i].repressor, cell[i].delta) +
                                        gdelta(predictor_repressor[i], predictor_delta[i
                                        ])) * epsilon * z2_delta[i];
91      }
92      delete[] predictor_repressor;  delete[] predictor_delta;  delete[]
            z1_repressor; delete[] z2_delta; delete[] M;
93  }
94
95  double valor_absoluto (double x)
96  { if(x<0.0) return -x; else return x; }
97
98  void construir_matriz_conectividad(bool *M, celula *cell)
99  {
100     //En cada fila i, la matriz de conectividad dice que celulas se tocan con la
            celula i.
101     int i, j, last_index;
102
103     //Se busca primero el ultimo indice empleado en cell
104     last_index = get_first_void_index(cell);
105
106     for(i=0; i<last_index; i++)
107         for(j=0; j<last_index; j++)
108         {
109             if(i==j)    *(M + j*(last_index-1) + i) = 0;    //Una celula consigo
                    misma no interactua
110             else if (distancia_entre_celulas(cell[i], cell[j]) <= R_cutoff) *(M
                + j*(last_index-1) + i) = 1;    //Si dos celulas se ven, la
                matriz tiene un 1
```

```
111             else *(M + j*(last_index-1) + i) = 0;
112         }
113 }
114
115 double promediar_delta_vecinos (celula *cell, int i_cell, bool *M)
116 {
117     int i, last_index, n_primeros_vecinos;
118     double delta_promedio;
119
120     delta_promedio=0.0;
121     n_primeros_vecinos=0;
122     last_index = get_first_void_index(cell);
123     for (i=0; i<last_index; i++)
124         if(*(M + i_cell*(last_index-1) + i)){
125             delta_promedio+=cell[i].delta;
126             n_primeros_vecinos++;}
127     if(n_primeros_vecinos == 0) return 0;
128     else return delta_promedio/(1.0*n_primeros_vecinos);
129 }
130
131 double promediar_delta_vecinos_vpredictor (celula *cell, double *delta_predictor
        , int i_cell, bool *M)
132 {
133     int i, n_primeros_vecinos, last_index;
134     double delta_promedio;
135
136     last_index = get_first_void_index(cell);
137     delta_promedio=0.0;
138     n_primeros_vecinos=0.0;
139     for (i=0; i<last_index; i++)
140         if(*(M + i_cell*(last_index-1) + i)){
141             delta_promedio+=delta_predictor[i];
142             n_primeros_vecinos++;}
143     if(n_primeros_vecinos == 0) return 0;
144     else return delta_promedio/(1.0*n_primeros_vecinos);
145 }
146
147 double distancia_entre_celulas(celula cell1, celula cell2)
148 {
149     return sqrt((cell1.x - cell2.x)*(cell1.x - cell2.x) + (cell1.y - cell2.y)*(
            cell1.y - cell2.y));
150 }
151
152 int get_first_void_index(celula *cell)   //Encuentra el primer indice en el que
        no hay ninguna celula
153 {
154     int last_index, i;
155     for (i=0; i<max_cells; i++)
156     {
157         if(cell[i].ID==0)
158         {
159             last_index=i;
160             i=max_cells+2;
```

```
161             }
162         }
163         return last_index;
164     }
```